ANGLE OF ARRIVAL DETECTION THROUGH ANALYSIS OF OPTICAL FIBER INTENSITY PATTERNS

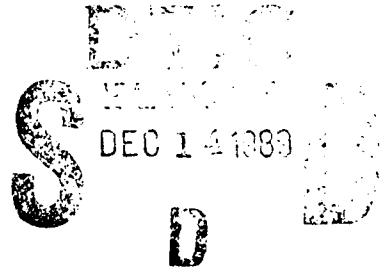
THESIS

John Wallace Welker
Captain, USAF

AFIT/GE/ENG/89D-57

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY
Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/89D-57



ANGLE OF ARRIVAL DETECTION THROUGH ANALYSIS OF OPTICAL FIBER INTENSITY PATTERNS

THESIS

John Wallace Welker
Captain, USAF

AFIT/GE/ENG/89D-57

AFIT/GE/ENG/89D-57

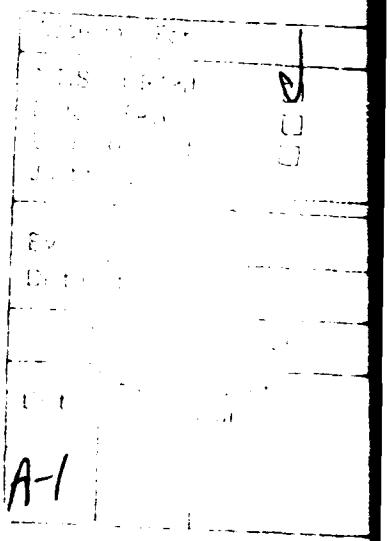
ANGLE OF ARRIVAL DETECTION THROUGH ANALYSIS OF OPTICAL FIBER INTENSITY PATTERNS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

John Wallace Welker, B.S.E.E.
Captain, USAF

December, 1989



Preface

This thesis demonstrated the feasibility of analyzing the output intensity patterns of an optical fiber to determine the angle of arrival(AOA) of incident laser radiation. Adequate information was contained in the low frequency Fourier components to determine AOA within 1°.

Many people contributed to this research. I would like to acknowledge the work of Capt Geno Cole of the Air Force Weapons Lab who first investigated this area of AOA detection in 1987. Further, I would like to mention the Chief Scientist Dr. Brendan B. Godfrey, Deputy Chief Scientists Lt Col Carl E. Oliver and Lt Col LaRell K. Smith, as well as Senior Technical Coordinator Richard Keppler, whose funding made this research possible. I'm also grateful to my thesis advisor, Dr. Steven Rogers, for demanding the best job possible, and the members of my thesis committee, Dr. Matthew Kabrisky and Lt Col James P. Mills. I would also like to thank Capt Dean Cambell for his technical support and Capt John Cline for his help incorporating images into LaTeX. I owe a great deal of gratitude to my parents for instilling in me the desire to achieve my goals and make the best out of the worst.

Most of all, I would like to express my love and appreciation to my children who sacrificed time with me for the last 18 months, and my wife whose words of encouragement and love in her heart gave me the strength and motivation to keep going.

Julianne, Jennifer, and Christopher; I'm coming home!

John Wallace Welker

Table of Contents

List of Figures

List of Tables

Abstract

This thesis examined the feasibility of using optical fibers for angle of arrival(AOA) detection. Specifically, a uniform amplitude plane wave was transmitted through a $3mm$ diameter, multimode, step-index, plastic fiber. Fiber lengths of $1in$, $2in$, and $3in$ were investigated. The output intensity pattern from each optical fiber was detected by a 512 x 512 resolution, charge injection device(CID) camera. The data array from the camera was preprocessed with a median filter, and normalized to the peak intensity. A fast Fourier transform converted the data to the frequency domain where the fundamental and first three harmonic frequencies were extracted. These 49 numbers represented components of a feature vector representing the pattern related to a specific AOA. Twenty- six template vectors, each representing a 1° increment from 0° to 25°, were created. Test vectors were compared to template vectors *using Euclidean distance*. The *minimum distance* classified the test vector AOA.

ANGLE OF ARRIVAL DETECTION THROUGH ANALYSIS OF OPTICAL FIBER INTENSITY PATTERNS

I. Introduction

The Air Force is currently investigating methods to determine the angle of arrival(AOA) of directed energy weapons, primarily lasers. An AOA detector has several applications. It could be used on an aircraft to alert the pilot to incident laser radiation which could blind crewmembers permanently or for as little as a few seconds, thereby putting their lives in jeopardy. Another application is for use on reconnaissance satellites to warn of laser radiation which could damage vital components. An AOA detector can be used in conjunction with power meters and spectrum analyzers to determine power and frequency of the source, which could be several hundred kilometers away. This information is vital to the intelligence community. This system must be compact, lightweight, shock resistant, highly reliable, and simple in construction, with an angular resolution of 1 degree or less (9:7). This thesis will investigate a solution to this problem through the use of a passive (acted upon) detector. The output intensity pattern from a large diameter optical fiber, specifically a $3mm$ diameter, multimode, plastic fiber is analyzed to determine the AOA. Since this detector will use short fibers(i.e. $8cm$ or less) and will contain no moving parts, it will meet the previously mentioned requirements.

1.1 Approach and Scope

The initial effort in this thesis is to research the feasibility of AOA systems using optical fibers. This includes a theoretical determination of the intensity profile

expected at the output of the optical fiber using geometrical optics. The second part of this research is experimental. A uniform amplitude plane wave will illuminate a short segment of $3mm$ diameter plastic fiber with the output impinging on a 512 x 512 charge injection device(CID) camera. Since this detector is to determine AOA of laser input at a substantial distance from the source, the input is assumed to be a uniform amplitude plane wave. The data is then reduced to a 256 x 256 array, smoothed through a median filter operation, and then normalized based on the highest intensity value. Following this, the results are Fourier transformed. This frequency domain processing involves extraction of the fundamental frequency and the first three harmonics to form a feature vector for each angle of incidence. Finally these feature vectors are compared, using Minkowski distance, to random angle test vectors in order to determine AOA of the test samples. This process is outlined in figure 1.

1.3 Organization

This thesis begins with a look at two AOA detectors utilizing optical fibers. It then theoretically investigates the propagation of light through an optical fiber to explain the cardioid output pattern seen in short fibers. Chapter 3 describes the laboratory setup, data acquisition, and processing. Results are discussed in Chapter 4 with conclusions and recommendations in chapter 5.

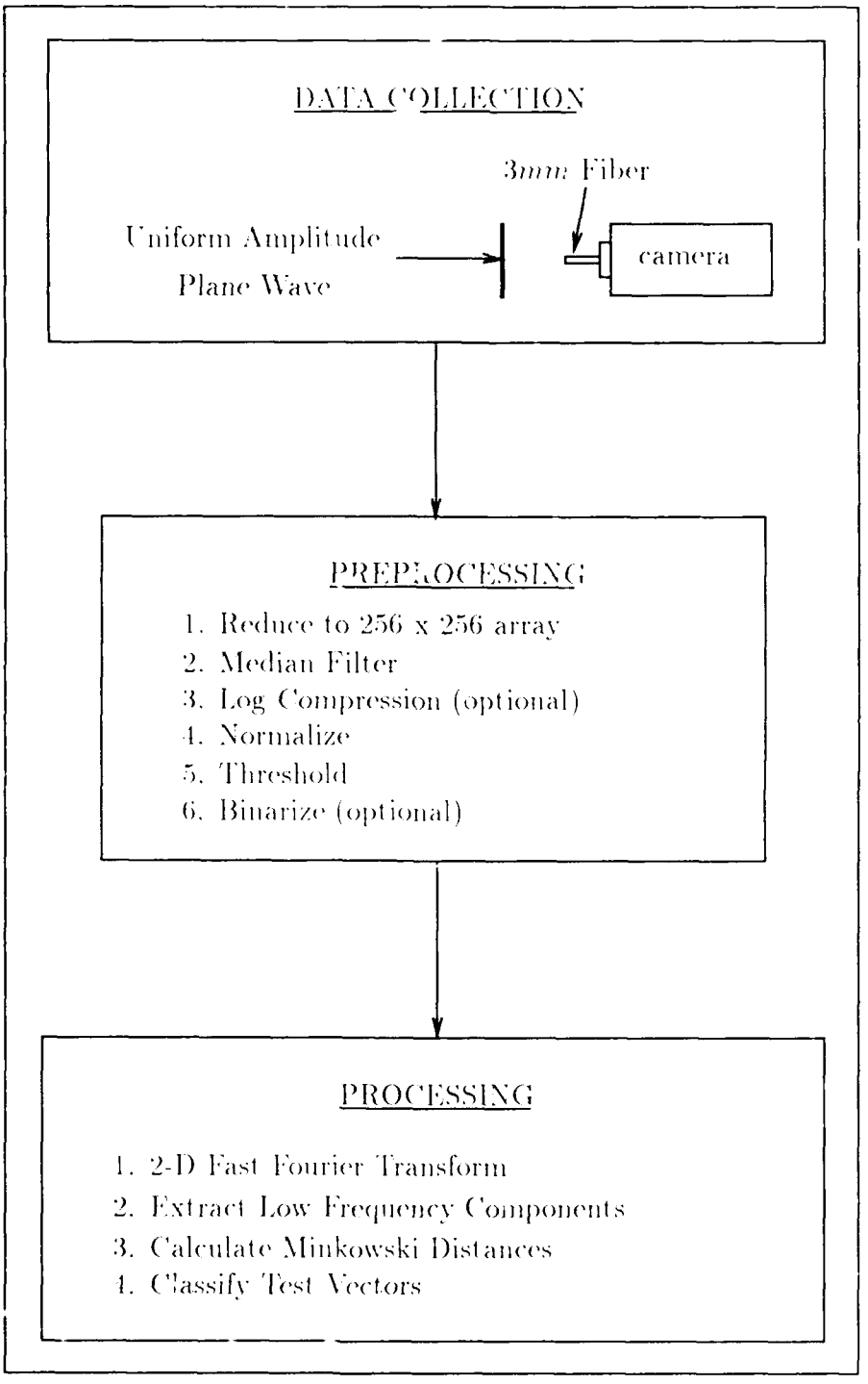


Figure 1. Block Diagram of AOA Process

II. Summary of Current Knowledge

Various systems have been developed in recent years to determine AOA of light energy. One such system uses a quadrant detector to give enough information to allow armored vehicle crews to home in on the source using the vehicle's night sight (5:1). Another system utilizes interferometry techniques to determine not only AOA of the incident laser radiation but also its wavelength (5:1). A third device uses the geometry of a triangular cube reflector with photodetectors aligned with the three sides to determine AOA (5:4-10). There are only two AOA systems using optical fibers (9, 10).

2.1 Optical Fiber Detectors

The first system was investigated in 1985 at Wright-Patterson AFB, OH. This system utilized an array of lenses mounted on a hemispherical dome. Each lens was separated by 22.5 degrees allowing overlapping fields of view(FOV). In other words, each lens saw a small portion of what its adjacent lens saw. This resulted in a total of 81 lens apertures. Each lens focused light energy into a collection of optical fibers of different lengths. Each bundle of fibers terminated onto a photodetector. Two different types of photodetectors were used. Silicon photodetectors were used to detect laser wavelengths from $0.6\mu m$ to $1.0\mu m$ while germanium photodetectors operated in the $1.1\mu m$ to $1.9\mu m$ spectral region, commonly known as the infrared(IR). The photodetector signals were used to excite underdamped circuits (10:2). Due to the overlap of the FOV of adjacent lenses, a mathematical relationship between the zero crossings of underdamped circuit oscillations and the optical centroid of aperture illumination was exploited. Signal-to-noise was altered as necessary to vary detection probability and false alarm rate (10:6). Tests were conducted with varying degrees and types of background lighting. Results showed the background lighting had negligible effect on AOA detection of the laser light (10:33-35). Angle of arrival

accuracy of ± 5 degrees was obtained (10:40). This is good, *but it is desired to have* an accuracy of 1 degree or less (9:7).

The second system is the subject of current research by a small firm based in Menlo Park, California and hopes to obtain the needed accuracy (9:23). This system is composed of several hundred apertures on the surface of a hemispherical dome. Each aperture has a 20 degree FOV and does not require lenses like the previous system (9:5). This system uses optical fibers to transmit the intensity incident on the apertures to individual silicon photodetectors. A processor using an algorithm similar to that used in Radar Warning Receivers performs the necessary comparisons of intensities from the irradiated fibers to determine AOA (9:7). It accomplishes this by utilizing a look-up table. This table was created by illuminating the hemisphere from different directions, performing the aforementioned signal analysis, and storing the results in memory. In actual use, the system will correlate the intensity of incident laser energy to the test patterns stored in memory. AOA of the laser radiation can by determined from this correlation (9:33-45). In order to achieve an overall FOV of 2π steradians(hemisphere), these photodetectors are spaced every 9 degrees, once again establishing overlapping FOV (9:8). This overlap is necessary so that the entire 2π steradian field is detected. It also insures that each sensor is not independent but interconnected so that the data received can be combined and correlated in an effective manner (9:18-23). The optical fibers in these systems are strictly used to transmit the intensity from an aperture with a small FOV.

2.2 Light Propagation in an Optical Fiber

This research exploits the analysis of the intensity profiles of individual non imaging fibers to gain additional information at the output of the fiber. Captain Geno Cole was the first person to investigate this theory. This thesis will perform the experimental portion of Captain Cole's initial investigation (1). Non imaging implies that there is no lens system used to focus an image plane on the detector

array in the CID camera. In addition, multimode step-index fibers do not transmit images well, due to intermodal dispersion. A look at how light propagates down a fiber will help to better understand this phenomenon. Since the diameter of the fiber used was significantly larger than the wavelength of the incident laser radiation, geometrical optics (i.e. ray tracing) can be used to examine wave propagation within the fiber (7:13). For the analysis, incident light rays will be separated into two classes. The first class, meridional rays, are easy to track since they stay within the meridional plane. A meridional plane is any plane which contains the axis of symmetry of the fiber (core axis). Ideally, there is only one meridional ray for an incident plane wave. The second class, skew rays, represents all off-axis rays (7:23). Figure 2 displays the meridional ray passing through a fiber. Snell's law

$$n_i \sin\theta_i = n_t \sin\theta_t$$

is used to determine its path as it propagates down the fiber (7:17). As the ray contacts the entrance of the fiber, some of the energy is reflected and the remainder is transmitted. As the angle of incidence increases, less energy is transmitted into the fiber (4:106). The transmitted ray is bent towards the core axis since the index of refraction of the core is greater than that of air. This is the case of external reflection. In the case of external reflection, some of the energy is always transmitted. As the ray contacts the core-cladding interface it is once again reflected, however, this time the ray is contacting the surface of a less optically dense medium (i.e. lower index of refraction) which represents the case of internal reflection. Thus the ray is bent away from the surface normal. The point of contact can be modeled as a plane tangent to the point of contact which can be called the reflection plane. As the angle ϕ decreases, α increases until it equals 90 degrees and the transmitted ray is parallel to the core-cladding interface. This value of ϕ is known as the critical angle, ϕ_{crit}. For any value of ϕ less than ϕ_{crit}, the ray will experience total internal reflection. Ideally this means that none of the ray is transmitted into the cladding,

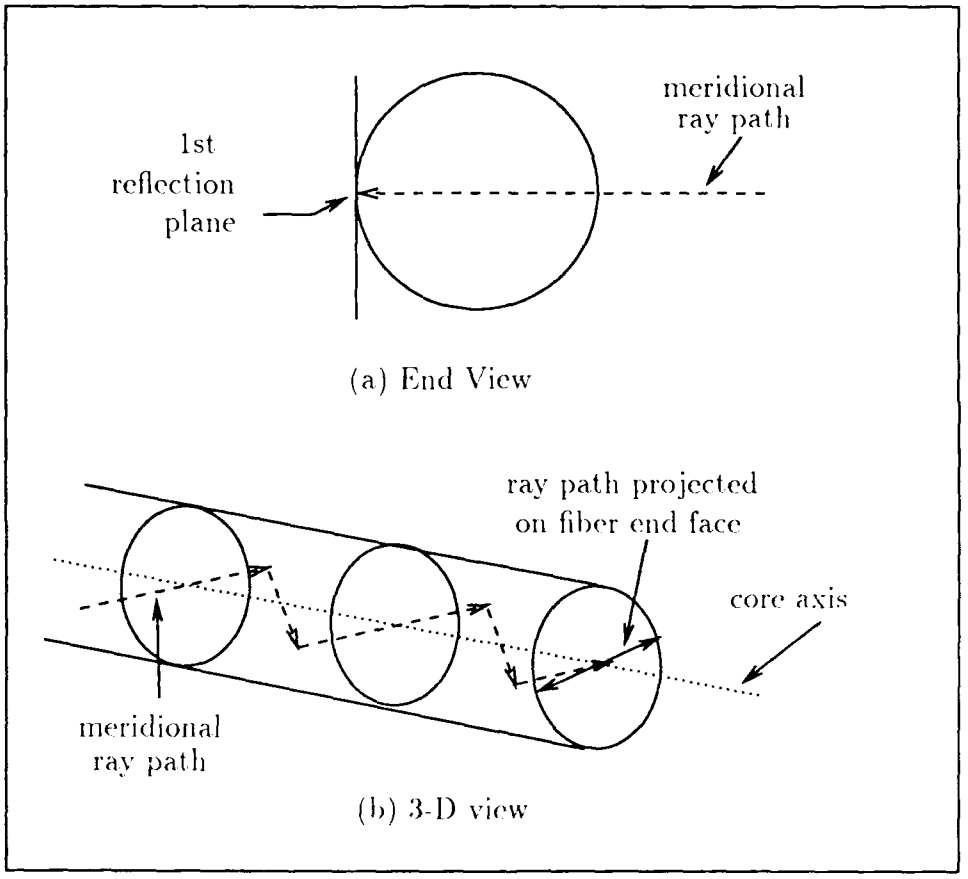


Figure 2. Path of Meridional Ray

but in reality some minimal loss always occurs in the form of a surface wave or evanescent wave (4:106,107). The ray continues down the fiber experiencing internal reflection at each core-cladding interface until it reaches the end of the fiber. The ray contacts the end of the fiber at an angle of ϕ degrees and once again experiences internal reflection since $n_{air} < n_{core}$. Thus the ray upon exiting the fiber will be bent away from the core axis. Figure 3 presents a graphical representation of this angular dependence (7:17-24).

The maximum entrance angle which allows total internal reflection can be calculated by Snell's law. Using figure 3 (b) as a reference:

$$n_{core} \sin (90° - \phi_{crit}) = n_{clad} \sin \alpha \tag{1}$$

where $\alpha = 90°$. Thus:

$$\sin (90° - \phi_{crit}) = \frac{n_{clad}}{n_{core}} = \cos \phi_{crit} \tag{2}$$

for the core-cladding interface.

At the fiber entrance:

$$n_{air} \sin \theta_{max} = n_{core} \sin \phi_{crit} \tag{3}$$

and since:

$$\sin \phi_{crit} = \left(1 - \cos^2 \phi_{crit}\right)^{\frac{1}{2}} \tag{4}$$

equation 3 becomes:

$$n_{air} \sin \theta_{max} = n_{core} \left(1 - \cos^2 \phi_{crit}\right)^{\frac{1}{2}} \tag{5}$$

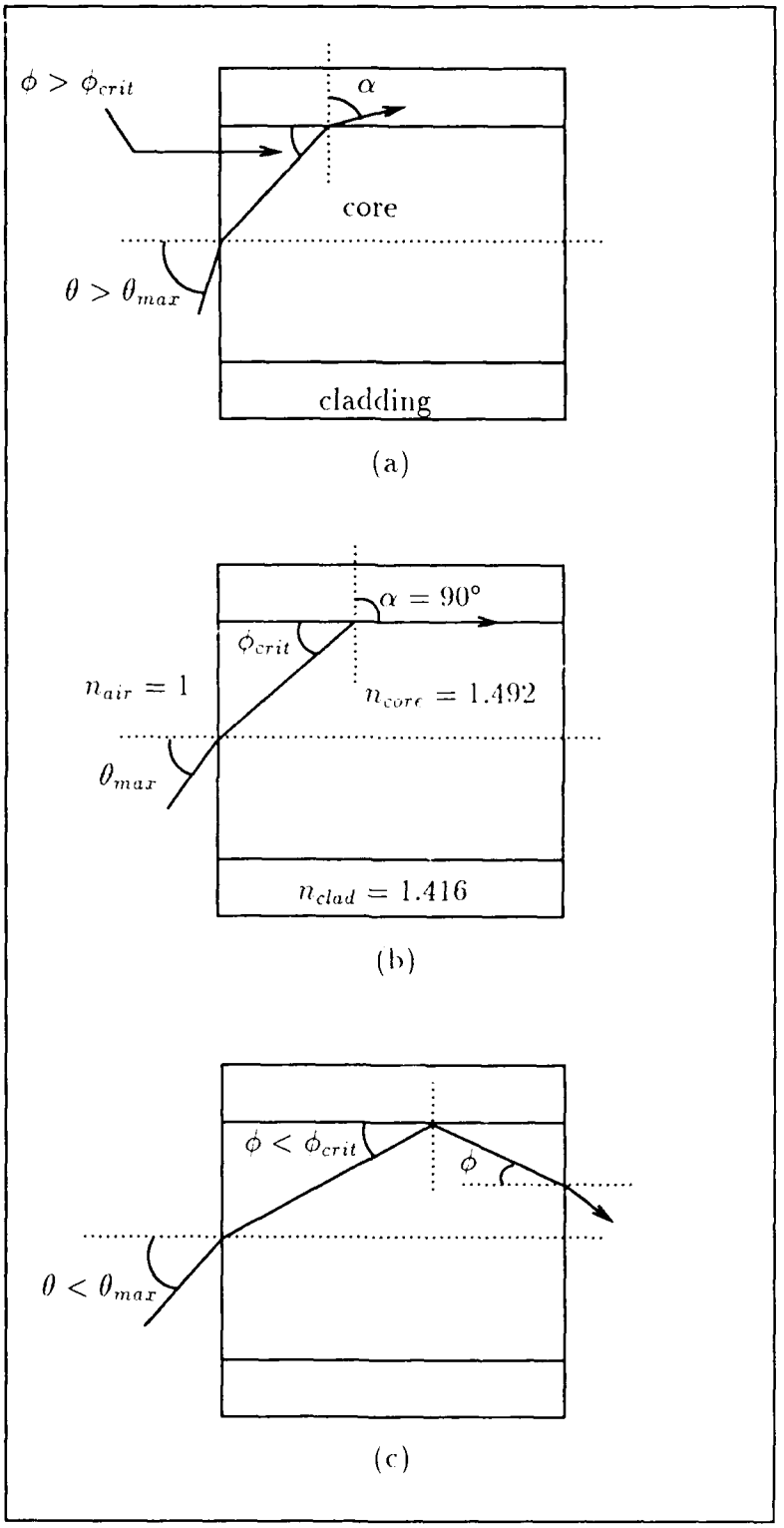


Figure 3. Relation of Entrance Angle to Critical Angle

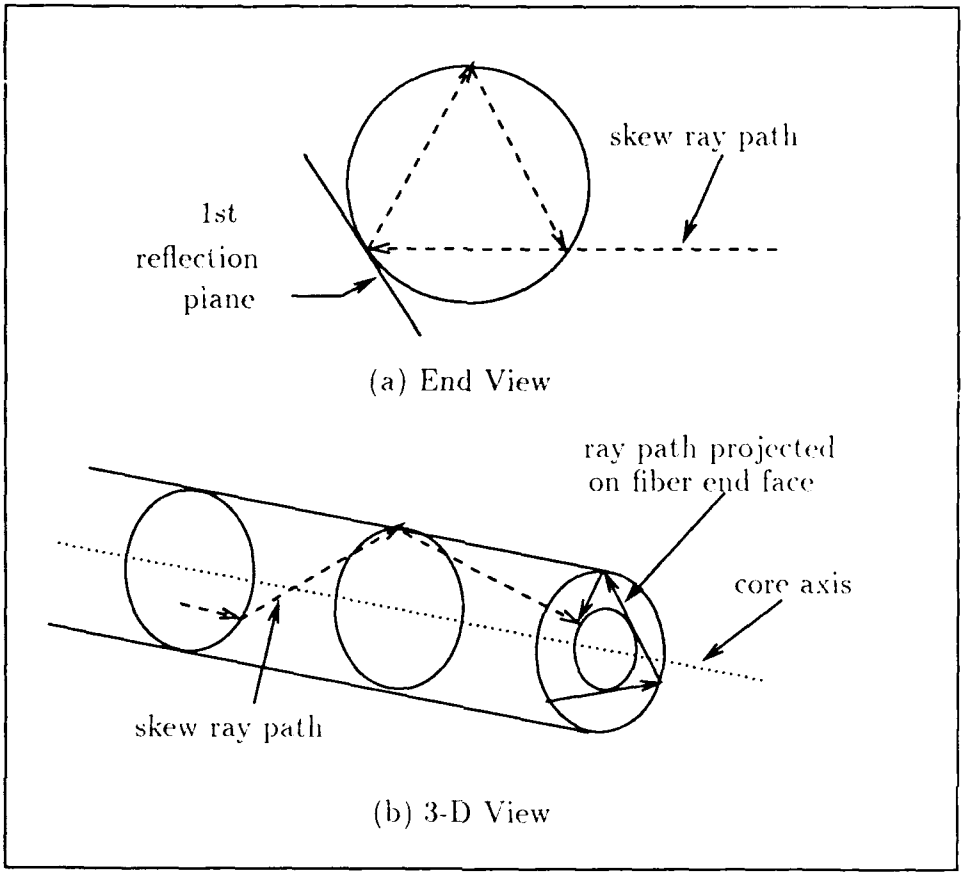


Figure 4. Path of Skew Ray

Substituting equation 2 into equation 5 yields:

$$n_{air} \sin \theta_{max} = n_{core} \left(1 - \frac{n_{clad}^2}{n_{core}^2}\right)^{\frac{1}{2}} \tag{6}$$

Since $n_{air} = 1$, equation 6 reduces to:

$$\theta_{max} = \sin^{-1} \left(n_{core}^2 - n_{clad}^2\right)^{\frac{1}{2}} \tag{7}$$

For the fiber used in this research, $n_{core} = 1.492$ and $n_{clad} = 1.416$ resulting in a maximum AOA of approximately 28° to achieve the critical angle at the core-cladding interface(7:24,25). Figure 4 illustrates the path of a skew ray. This ray experiences

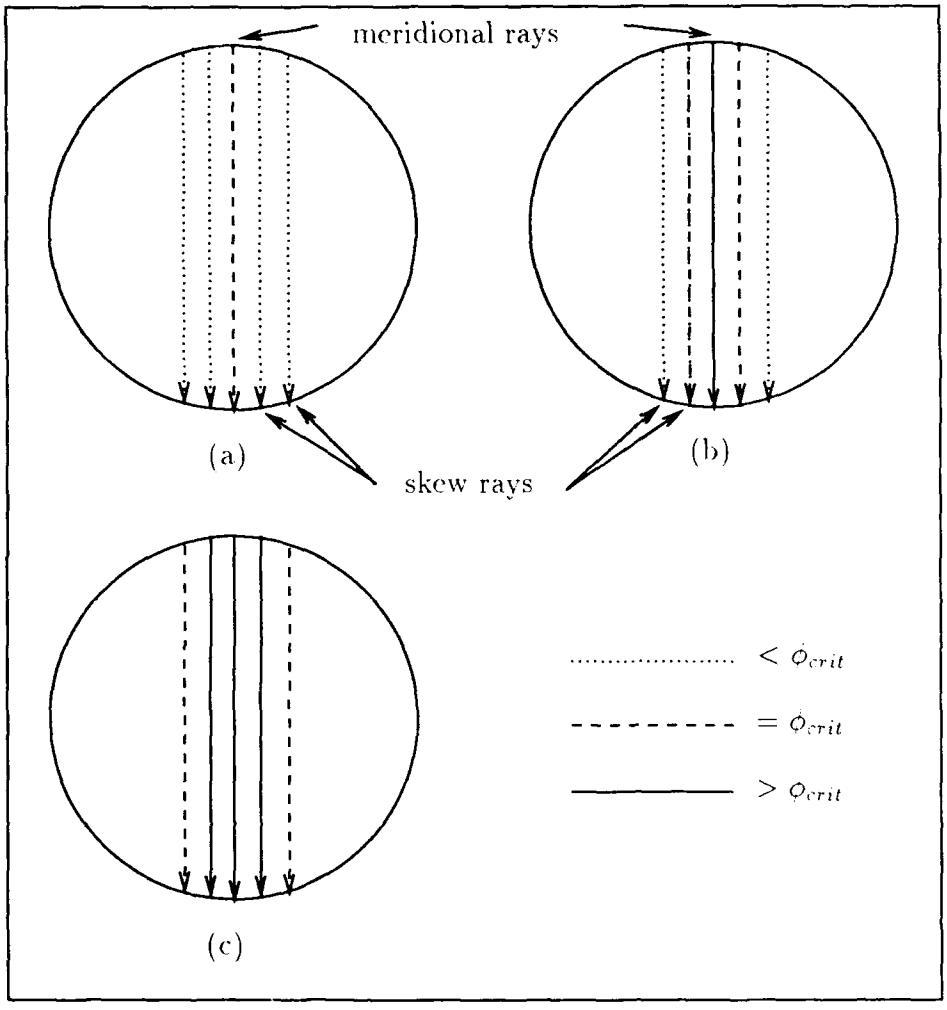


Figure 5. Critical Angle Comparison of Meridional and Skew Rays

the same reflection and refraction as the meridional ray as it contacts the entrance to the fiber since it is part of the same uniform amplitude plane wave. It continues to travel parallel to the meridional ray until it reaches the first core-cladding interface. At this interface it reflects differently than the meridional ray since its reflection plane is oriented differently, as shown in figure 4 (a). Figure 4 (b) shows how this ray continues its propagation along the fiber.

Figure 5 is a magnified end view of a small sample of adjacent rays composing the incident uniform amplitude plane wave. The center ray is the meridional ray.

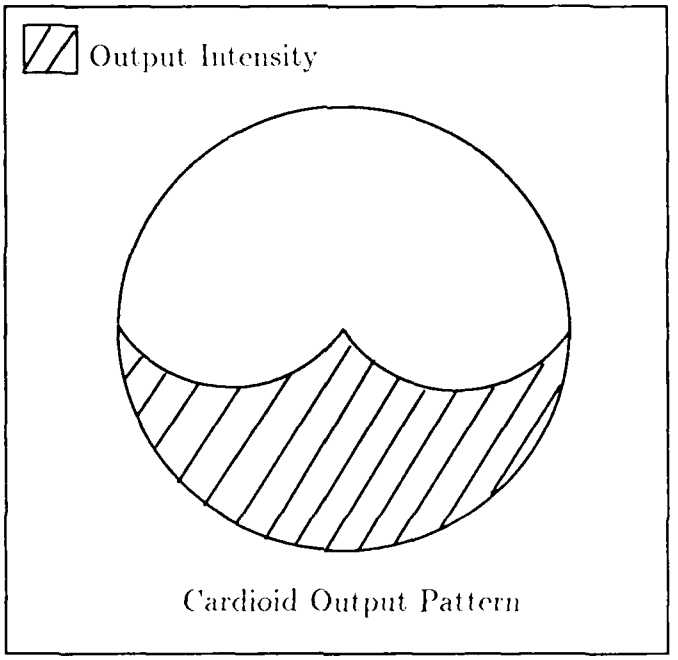


Figure 6. Cardioid Output Intensity Pattern of an Optical Fiber

All other rays are skew rays. These views only show the rays as they enter the fiber and contact the core-cladding interface for the first time. Continuing propagation down the fiber is not shown. Figure 5 (a) shows that if the incoming meridional ray is at the critical angle, the skew rays are less than the critical angle since ϕ , in figure 3, is now divided into a transverse as well as a longitudinal component. Thus the only ray at the critical angle ϕ_{crit} is the meridional ray. If the angle of incidence of the plane wave is increased slightly, the meridional ray as well as the immediately adjacent skew rays are $\geq \phi_{crit}$. This is shown in figure 5 (b). Figure 5 (c) shows that as the angle of incidence continues to increase, progression continues outward from the center with more and more skew rays reflecting at angles $> \phi_{crit}$. Figure 6 displays the output of a large diameter, multimode, step index fiber. The angle of incidence is much less than the critical angle and the fiber is only long enough to allow one reflection at the core- cladding interface. As expected, the meridional ray passes through the end of the fiber at the highest longitudinal angle while the

skew rays share their longitudinal angle with a transverse angle, thus reducing the longitudinal angle. This results in the cardioid pattern seen in figure 6. This cardioid pattern was first predicted by a ray tracing program, *sensor.pas*, written in Turbo Pascal by J. Scott Hunt, Cadet, United States Air Force Academy. Cadet Hunt wrote this program earlier this year during a temporary duty assignment at the Air Force Weapons Lab. As the angle of incidence increases, eventually a second reflection is allowed to take place in the fiber. During this time the cardioid pattern moves through the center of the output to the other side. As the angle of incidence increases beyond the critical angle, the cardioid pattern continues to move as before but experiences a loss of intensity due to less transmission (higher reflection) at the entrance of the fiber and loss of total internal reflection within the fiber.

2.3 Preprocessing

To remove noise spikes from images and smooth the data, a median filtering operation can be performed. The process is simple. For an N x N median filter the intensities in the first N rows and N columns of the image array are read. These N^2 numbers are then sorted lowest to highest. The middle number represents the median value and it is placed in the center of the N x N array. A new image array is created and begins with the N x N array. Going back to the original array, the next set of N^2 numbers is still taken from the first N rows but shifted one column to the right, thereby using intensities from columns one to $N + 1$. The process continues, storing the median values in the new array. This type of filter will not affect step changes in intensity but will adjust a noisy pixel to a value representative of its neighbors. An $(N + M)$ x $(N + M)$ array will smooth more than an N x N array, however, it may take significantly longer to process (6).

2.4 *Data Reduction*

In order to increase processing speed, some form of data reduction must be employed. Information about the shape and magnitude of the output of an optical fiber contained in a large array of pixels(i.e. 256 x 256) can be reduced by Fourier Transform(FT) methods. A FT decomposes a waveshape into an infinitely countable set of sines and cosines. The FT equation is (2:313):

$$F(u,v) = \mathcal{F}\{f(x,y)\} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-j2\pi(xu+yv)}dxdy \tag{8}$$

The general shape of the pattern is contained in the lower frequencies of the FT while more detailed information, such as sharp corners, is contained in the higher frequency components of the FT (3:262,263). As an example, the square wave in figure 7 (a) can be represented by a sum of sines and cosines. In this specific example the coefficients of the cosine terms as well as the even harmonic sine terms are zero. Thus only odd harmonic sine waves are needed to represent this waveform. In the general case, however, all coefficients will have non zero values. As shown in figure 7 (b), the general shape of the square wave can be achieved using only the first few lower frequency components. In the case of a stored digital array of intensities, the array can be converted to the frequency domain by means of a fast Fourier transform(FFT). Although the input array may be real numbers only, the output of the FFT will be a real array and an imaginary array. These arrays are replaced with a single magnitude array by combining each array element through the formula:

$$Magnitude = \sqrt{(Re)^2 + (Im)^2}$$

This magnitude array contains the lower frequency components in each of the four corners. In order to place the center frequency at the center of the array surrounded by the lower harmonics with frequency increasing outward from the center, the first and third as well as the second and fourth quadrants must be swapped, commonly

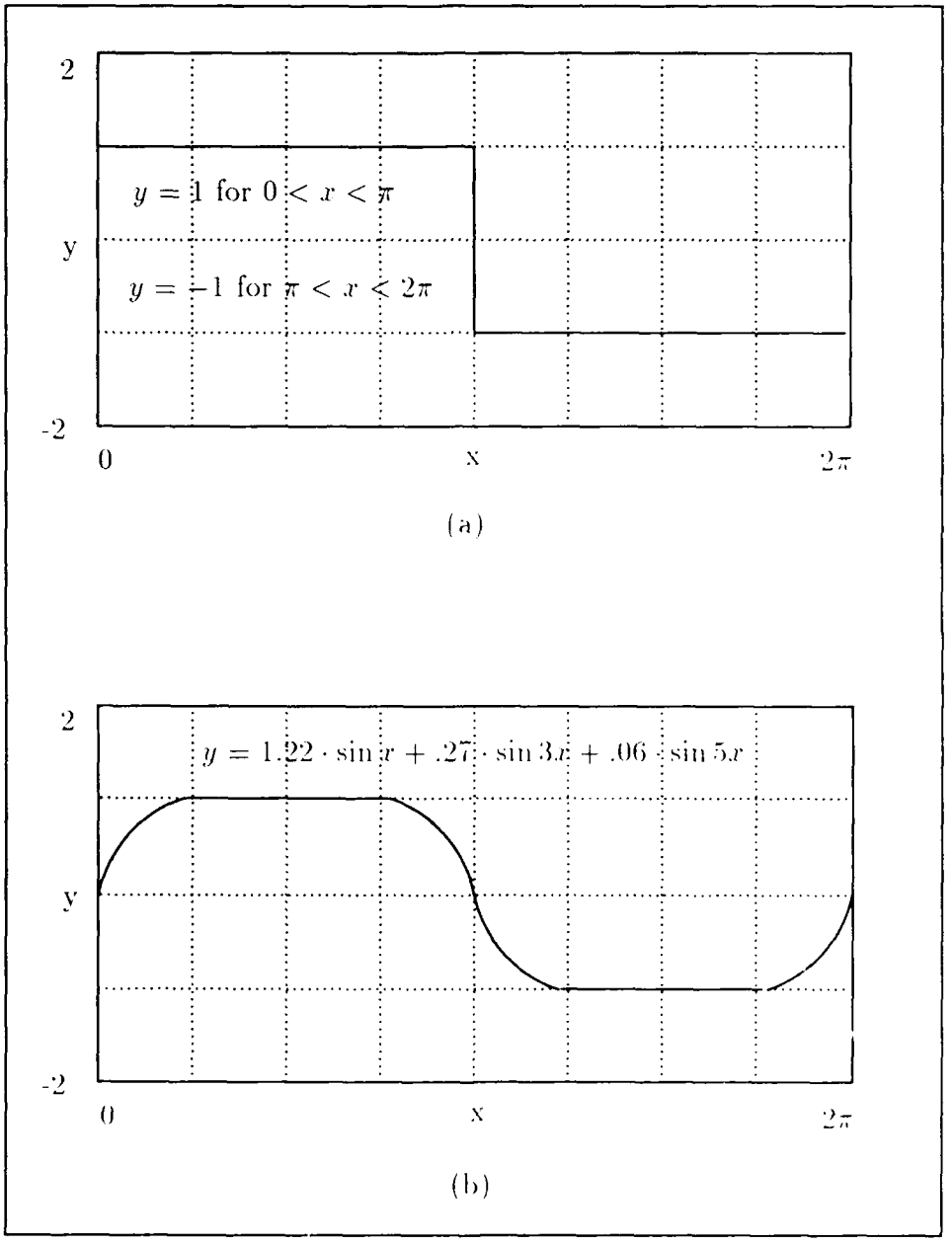


Figure 7. Low Frequency Representation of Square Wave

called quadrant swapping. The fundamental frequency is now in the center. The second harmonic is composed of the eight numbers surrounding the center. The third harmonic consists of the sixteen numbers surrounding the second harmonic. This process continues outward to the edge of the array. Therefore, information regarding the general shape of a large array can be stored in the first few harmonics of a magnitude array of spatial frequencies.

2.5 Comparison of Images

Since information concerning the shape of an image can be stored in the lower harmonics of spatial frequencies, these components can represent the components of a vector commonly called a feature vector. This results in a vector of N components for each image. Comparing the images is just a matter of calculating distances between N-dimensional feature vectors. This is done using the Minkowski distance:

$$M_N = \sqrt[N]{\sum_{i=1}^{M} | x_i - y_i |^N}$$

where N is an integer. For $N = 1$, M_N is called *taxi* distance and for $N = 2$ it is known as *Euclidean* distance. A 2-D example is shown in figure 8.

The tips of the two vectors lie at (2,3) and (5,1). Taxi distance equals:

$$M_1 = (5 - 2) + (1 - 3) = 4$$

and the Euclidean distance equals:

$$M_2 = \sqrt{(5 - 2)^2 + (1 - 3)^2} = \sqrt{10}$$

If the distance in N Space between M vectors is large enough to remain separated when noise is added to the system, the Minkowski distance can be used for pattern recognition. This is accomplished by assigning a template vector to each known

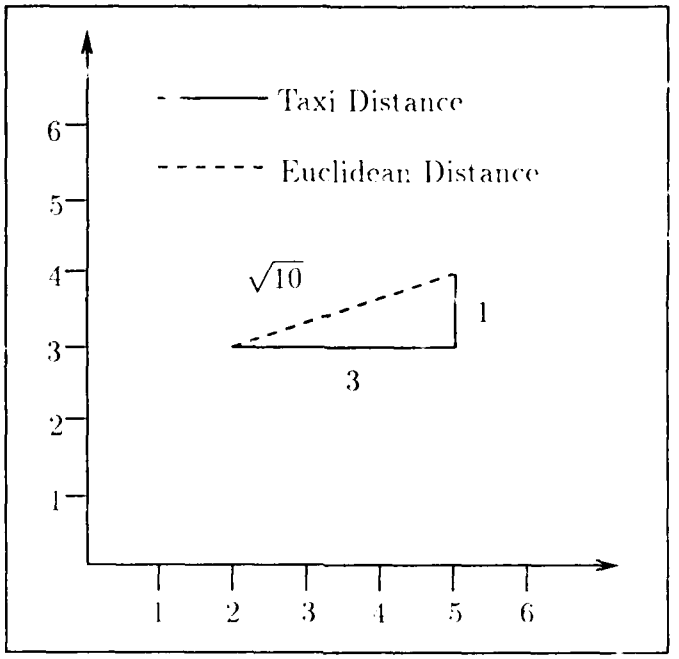


Figure 8. Taxi Distance and Euclidean Distance

shape and calculating the distance to a test vector representing an unknown shape. The closest template vector classifies the test vector.

2.6 Summary

This chapter examined two optical fiber systems which use the power from several fibers to determine AOA. Propagation of light radiation through a fiber was presented in order to understand the cardioid output from a short fiber. Then smoothing, data reduction, and a pattern recognition technique using feature vectors were reviewed in an effort to demonstrate feasibility of determining AOA through analysis of the intensity pattern from the output of an optical fiber. Chapter 3 details the laboratory setup, equipment used, and processing software used.

III. Experimental Approach

The experimental portion of this thesis used a Spiricon frame grabber, optical bench, and software written in 'Turbo-C' and 'ADA' to perform the necessary processing of the data. All 'Turbo C' programs were written by the author. The 'ADA' program used for the fast Fourier transform(FFT) was supplied by George Letourneau, GE-89D (8). This chapter examines the design of the optical bench, data collection, and the processing of that data. The reader is assumed to be familiar with use of an IBM or compatible personal computer(PC) using the MS-DOS operating system.

3.1 Spiricon Frame Grabber

This experiment required the necessary hardware and software to analyze the output intensity pattern of an optical fiber as it was rotated through several angles of incidence of incoming laser radiation. The Spiricon 2250 Laser Beam Diagnostics System was chosen since it provided high image resolution. This system was used to grab frames of data, store that data in a file, and display a 3-D plot of the image. The system consists of a processing card which uses two slots in an IBM compatible personal computer(PC), version 5.0 Laser Beam Diagnostic software, a CIDTECH 512 x 512 Charge Injection Device(CID) camera, a real-time display monitor, and the necessary connecting cables. As just mentioned, the 2250 system is advertised to work within an IBM AT, IBM 386, or IBM compatible. After much experimentation with various PC's, it was discovered that the 2250 apparently only works on an IBM AT. This may be due to an addressing problem within the Z-248's and IBM 386 in which the 2250 system failed to operate. Details regarding the operation of the Spiricon 2250 Laser Beam Diagnostic System can be found in appendix A.

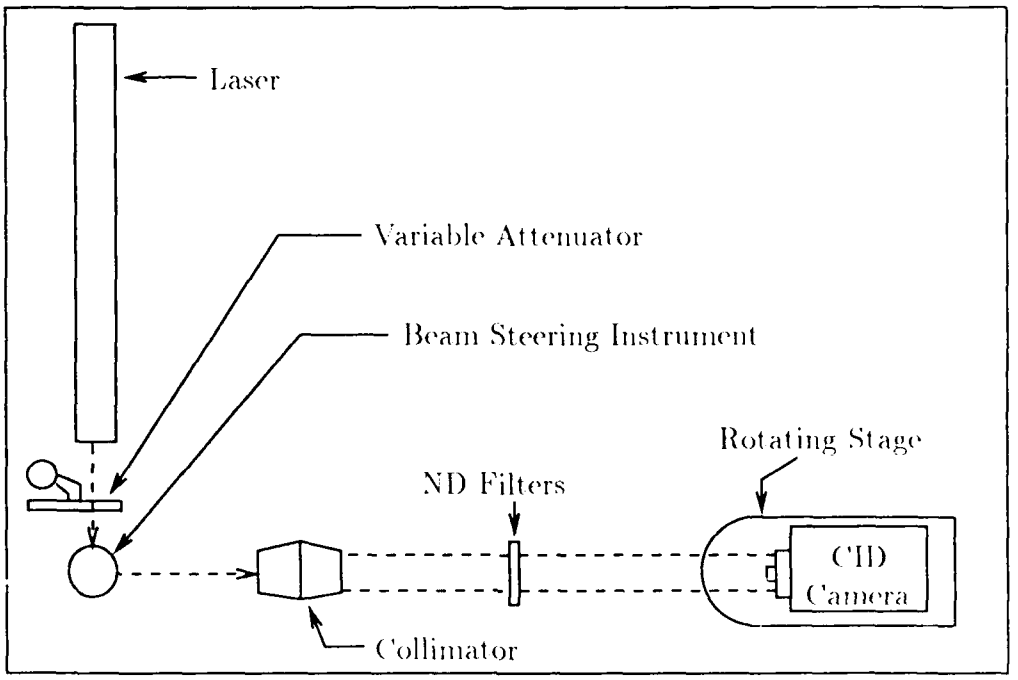


Figure 9. Optical Bench Layout

3.2 Design of Optical Bench

Since this was a new area of research, the optical bench had to be designed and built. This required the fabrication of several fixtures. The end result was a simple, reliable, and easy to use system for gathering data. Figure 9 details the optical bench design.

Since this detector will be used at large distances from the source of laser radiation, the input to the detector was assumed to be a uniform amplitude plane wave. A 50mW Newport Equipment Corporation(NEC) He-Ne continuous wave laser provided input to the system. The beam passed through a variable attenuator and onto a beam steering instrument. This reduced beam intensity and made the job of beam alignment significantly easier. Once the beam was aligned parallel with the top and one edge of the optical bench using a reference point on a movable screen, an NEC beam collimator using a $5\mu m$ pinhole was placed in the beam path.

To simplify the job of alignment, the collimator was bolted to a transverse, fine resolution, movable stage which was mounted on a laboratory jack. The output of the collimator was a $2.5cm$ diameter Gaussian profile. This was passed through a $3mm$ aperture aligned on the reference point. This aperture matched the size of the fiber and was used strictly to align the camera and fiber assembly. To facilitate the mounting of the fiber within the camera, special C-mount fixtures were fabricated with $3mm$ holes through the center allowing a snug fit of the fiber within the fixture. Three fixtures measuring $\frac{3}{4}in$, $1\frac{3}{4}in$, and $2\frac{3}{4}in$ were fabricated to facilitate three fibers in lengths of $1in$, $2in$, and $3in$ respectively. The fixtures were threaded outside to screw into the C-mount lens casing of the camera. This provided rigid coupling of the fiber and camera. Each fiber protruded $1mm$ from the end of the C-mount fixture inside the camera. This insured $1mm$ separation between the end of the fiber and the glass plate covering the CID detector array. Since the glass plate was $1.5mm$ thick, the distance between the end of each fiber and the detector was $2.5mm$. To simulate various angles of arrival(AOA), the camera assembly was placed on a rotating stage. This required another fixture to be designed and fabricated to mount the camera to the rotation stage. This fixture allowed the camera to be translated in a radial direction to facilitate alignment of the end of the fiber with the rotational axis while keeping lateral movement to a minimum. The camera and rotation stage was mounted to a transverse, fine resolution, movable stage. Spacers were used to adjust height since another laboratory jack was not available. The camera was aligned by centering the fiber on the $3mm$ beam passing through the aperture. To center the end of the fiber on the axis of rotation, the camera was turned on(live mode) and then rotated approximately 25 degrees from boresight. The camera was then translated radially for maximum intensity and locked down. This is shown in figure 10.

The $3mm$ aperture was then replaced by 2 neutral density filters with a combined optical density(OD) of 1.2(i.e. $10^{1.2}$). In addition the variable attenuator had

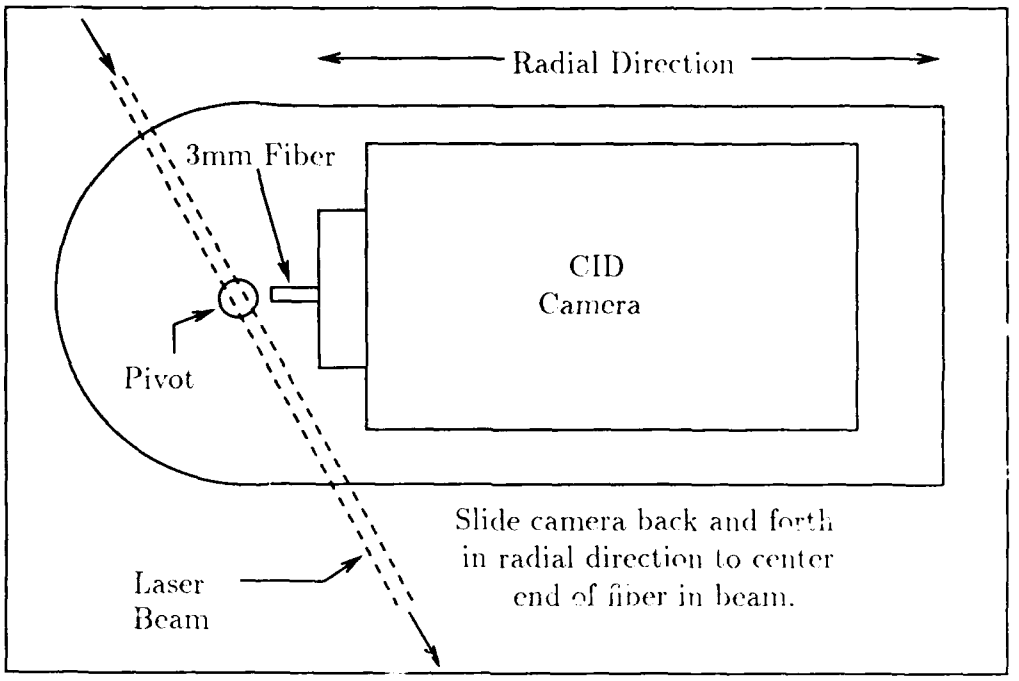


Figure 10. Centering End of Fiber on Axis of Rotation

to be adjusted to maximum attenuation(OD = 2) to keep the camera saturation to a minimum. Since the $3mm$ optical fiber sampled a small portion of the $2.5cm$ Gaussian beam, a uniform amplitude plane wave was simulated. Since the camera array was sensitive to visible light, and no filtering was readily available, data was taken under conditions of very low ambient light.

3.3 Preparation of Fibers

Before the fibers could be used for this research, the ends had to be polished as flat and smooth as possible. The following procedure details this work. The polisher used was a Buehler Fibrmet dual platen polisher. The fiber was cut approximately $2.5mm$ oversize by etching one side of the fiber with a razor blade. The fiber was then grasped in both hands and snapped to achieve a rather clean break. The fiber was then inserted into an adapter plate mounted on the polisher. This adapter plate was specifically fabricated for the $3mm$ fiber used in this research. After the

fiber was properly mounted on the polisher, a $12\mu m$ grit aluminum oxide polishing disc(yellow) was placed on the right platen. These discs are color coded and range from $0.3\mu m$ to $12\mu m$ in surface texture. The polisher was started and a bead of fibrinet polishing extender was placed on the disc. After 15 minutes, the course disc had removed approximately $1mm$ of material from the end of the fiber. The end of the fiber was now flat but rough. At this time the $12\mu m$ disc was replaced with a $3\mu m$ silicon carbide disc(gray) and a $0.3\mu m$ aluminum oxide disc(white) was placed on the left platen of the polisher. A small bead of polishing extender was placed on each disc. Once again the fiber was placed against the right platen, this time for only 2 to 3 minutes. The fiber was then smooth enough to be finished on the left platen. This finishing step took anywhere from 6 to 10 minutes. The adapter, with the fiber still securely clamped, was then removed and the end of the fiber examined with a small magnifying lens. Removing the adapter and fiber rather than the fiber alone helped to insure that the end of the fiber would be reinstalled parallel to the surface of the polishing disc if further polishing was necessary. Two fibers were polished for each of the three lengths.

3.4 Gathering Template Data

With the camera in live mode, the camera fiber assembly was rotated to find boresight(i.e. zero degree AOA) on the fiber. The stage was then locked down and the angular measurement on the rotation stage noted. In this case, zero degree AOA on the fiber was $0° + 25'$ on the rotation stage.

Data from the $3in$ fiber was taken first since the longer fiber exhibited the greatest amount of change per degree of motion. This was expected due to the larger number of reflections imposed by the increased length. This frame was executed and stored in file 0DEG3. This represented a zero degree AOA on the $3in$ fiber. The stage was rotated one degree to the $1° + 25'$ mark on the rotational stage. This frame was executed and stored in file 1DEG3. It is important that the files are

named in this manner with no extension since these are the filenames recognized by the preprocessing software. This procedure was repeated through 25° resulting in a total of 26 template files. Data was then gathered for the $2in$ and $1in$ fibers. Those files were labeled *DEG2 and *DEG1 respectively where the * represents some arbitrary AOA. The angle of incidence was the only parameter allowed to vary during the initial data gathering portion of this research.

3.5 Preprocessing

To help determine the optimum preprocessing, a program was written which performed various operations on the data. These included median filtering, normalization based on highest intensity, log compression, location of peak intensity, and thresholding to name a few. This program called *log3x3.c* allowed direct manipulation of full size Spiricon files. Further details on this program can be found in appendix B. Through the use of this program various preprocessing schemes were tested and the results displayed on the monitors to observe the effects. These results were used to design the final preprocessing program *preproc.c*.

The program *preproc.c* first reduced the 512 x 512 Spiricon data to a 256 x 256 array. This was done for two reasons. First, because of the small size of the fiber relative to the sensor array, the fiber output intensity pattern fit within the center 256 x 256 pixels. Second, and most importantly, this reduced the 263,052 byte Spiricon file to 65,536 bytes. This reduced file was strictly data with no header or parameter information. Next, the data was median filtered. This median filter differed from the filter described in chapter 2 in that it placed the median value into the center pixel of the array currently being filtered instead of creating a new array. Therefore each median sort was performed using a previously sorted median value. This was named a recursive median filter. Through the use of the previously mentioned *log3x3.c* program, this was discovered to be more efficient than the median filter described in chapter 2 . This process helped remove the significantly higher intensities introduced

by overly sensitive pixels in the CID camera array. This operation thus removed a great deal of unwanted noise while preserving the shape of the pattern.

Following the filtering operation, the data may or may not be converted to a compressed logarithmic scale. This helps increase the low intensities with little effect on the higher value intensities. This option was not selected during this research effort. The program next entered a normalization routine which scaled the data such that the highest intensity in the file was set to 255. This took advantage of the full range of data values allowed by Spiricon. Following this, the data was thresholded. This option allowed the user to not only choose a threshold value but also select whether the data should be binarized. In other words, everything below the threshold value is set to zero, but if the binary option is selected, then everything above the threshold is set to 255. Otherwise, those values above the threshold are left untouched.

To help select a threshold value, a histogram option can be chosen in the beginning of the program. This option places a histogram of the normalized data for each file into the file TEMPHIST.DOC or TESTHIST.DOC depending on whether template or test data is being preprocessed. These files also contain the sum of intensities and the peak value before and after normalization, as well as the coordinates of the peak intensity. Figure 11 illustrates this histogram. The output files are appended with a .256 extension and placed in the PROCDATA subdirectory of SPIR. This program will process multiple files without user intervention. Further information on this program can be found in appendix C.

3.6 Processing

Information concerning the general shape of an image is carried in the low frequency Fourier components (3:262,263). Thus, an attempt was made to write an FFT program to run on the PC that could process a 256 x 256 unsigned byte array. Two problems were encountered. The first involved the conversion of each

HISTOGRAM OF DATA
(Value in parentheses is value on spiricon plot)

0.00	percent of the values lie between	0(0)	and	15(127)
3.47	percent of the values lie between	16(128)	and	31(255)
5.92	percent of the values lie between	32(256)	and	47(382)
16.98	percent of the values lie between	48(383)	and	63(510)
27.01	percent of the values lie between	64(511)	and	79(637)
12.07	percent of the values lie between	80(638)	and	95(765)
13.92	percent of the values lie between	96(766)	and	111(892)
9.34	percent of the values lie between	112(893)	and	127(1020)
3.34	percent of the values lie between	128(1021)	and	143(1147)
3.27	percent of the values lie between	144(1148)	and	159(1275)
1.72	percent of the values lie between	160(1276)	and	175(1402)
1.53	percent of the values lie between	176(1403)	and	191(1530)
0.88	percent of the values lie between	192(1531)	and	207(1657)
0.25	percent of the values lie between	208(1658)	and	223(1785)
0.21	percent of the values lie between	224(1786)	and	239(1912)
0.1	percent of the values lie between	240(1913)	and	255(2040)

Figure 11. Histogram of Normalized Data

intensity value from one byte (unsigned) to eight bytes(signed), thereby allowing double precision floating point calculations in computing the FFT. This process increased the size of each file to 524,288 bytes. This added to the second problem of limited memory. The IBM PC allocates Random Access Memory(RAM) in 64Kbyte segments. Significant time was spent trying to perform the FFT on this large data file within the memory constraints imposed by the PC. Eventually this was overcome, but the results from the FFT were erroneous. During the troubleshooting, it was discovered that George Letourneau had a working version of an FFT written in ADA which ran on the VAX 11/780 using the VMS operating system (8). This program accepted a 256 x 256 array, unsigned byte format, and returned a 256 x 256 unsigned byte array representing the magnitude of the FFT. This program performed all the necessary quadrant swapping to center the frequency components.

To use this program, transfer the data files from the PC to the mainframe in the following manner. First copy the data files to floppy disks. Take these disks to room 243 and install in drive B of a PC containing the File Transfer Program(FTP). Drive B will handle double density and high density disks. Type FTP followed by the name of the computer which has the FFT program(i.e. FTP i780a). At the FTP prompt type BINARY since data and not text is being transferred. Now type MPUT b:*.256 for a multiple transfer, and all files will be transferred to the mainframe. Once transfer is complete, type BYE to return to DOS. Logon to the mainframe from an appropriate terminal and type RUN 2DFT. Type in the filename of the file to be processed at the prompt. Upon completion of the FFT, the program will ask for the name of an output file. Type the original filename with an FT prefix. For example, if the input file is 0DEG3.256, the output file will be FT0DEG3.256. After the filename is selected, the program will continue to perform a Hough transform, so type CTRL Y to exit the program and restart it for the next file. After all files have been Fourier transformed, transfer the files back to drive B on the PC by using MGET instead of MPUT at the FTP prompt.

When this program was first used, some unexpected results were obtained. Most of the files had outputs of zero magnitude. It appeared that the FFT program required binary input since the only files which yielded expected results were those with intensities of 255. All files should have been normalized to 255, but a rounding error resulted in a normalized peak value of 254. However, five of the 26 template files had peak intensities of 255 when the frame was executed and therefore did not require normalization. The rounding error was corrected, and the files were reprocessed. This time the files were binarized to insure the entire image was Fourier transformed. The threshold was set at 64 to reduce noise. These files yielded good results. After further investigation it was confirmed that the FFT program did not recognize values below 255. The FFT program was corrected and better results were obtained. These results, as well as problems with the noise reduction threshold, are discussed in chapters 4 and 5.

To extract the fundamental and first three harmonics of the FFT, a program was written called *template.c*. This program accepted input files with the FT prefix. The files were read in one at a time and values in the center 7 x 7 array, which represented the desired low frequency components, were written into each row of the file TEMPLATE.DAT in the TEMPLATE directory. This resulted in a template file of 26 rows and 49 columns. Each row represented a 49 component feature vector containing the shape information of the original image. A program called *distance.c* was written to calculate the separation between each of the 26 vectors in 49-D space to insure adequate distance between the vectors, thus allowing less chance of error when classifying a noisy test vector. The distance between each template vector and its nearest neighbor was placed in the file TEMPDIST.DOC. Further information on these two programs can be found in appendix D and appendix E respectively.

3.7 Classification of Test Vectors

Gathering test data was the next step. The camera was rotated to an arbitrary angle and locked down. The AOA was taken from the rotational stage and written down for future reference. This frame was executed and stored in file 0TST*. The 0 in this case represented the 0 number test vector, not the number of degrees, since our objective was to classify this file using the previously created template file to determine the AOA. The * represented the fiber length, similar to the previously mentioned template files. This process was repeated 8 more times resulting in 9 test vectors numbered 0 through 8. These files were processed in the same manner as the template files except for a minor difference. This time the *template.c* program created a TEST.DAT file containing the 9 test vectors and the *distance.c* program calculated the distances between the template vectors and each test vector. These distances were stored in TESTDIST.DOC. The minimum distance classified the test vector AOA. The results are presented in Chapter 4.

3.8 Software Verification Procedures

A significant portion of the time invested in this research was dedicated to software. This included not only programming but also verifying correct results. 'C' is not a very restrictive language, and many incorrectly written programs will still compile, producing incorrect results. Thus, verifying correct output at various stages of programming was necessary. The first program written, *log3x3.c*, contained several routines which were individually verified. Verification also took place anytime a change was made to the program.

Programming the median filter was a little tricky since the data arrays used in this thesis effort were to large to be stored in the PC memory. To overcome this problem, the data was segmented and no more than 100 rows were read in at one time. In order to correctly perform the median filtering operation, these segments had to overlap. A file named PLAY containing consecutively increasing

integers was created to verify the correct operation of the median filter, especially at the seams where the segments overlapped. Since a median filter will not affect consecutively increasing numbers, a correctly operating median filter would produce no change in this file. Once this small 16 x 16 file demonstrated correct operation of the median filter, the routine was changed to handle the 512 x 512 Spiricon arrays. The output of this larger array was then displayed on the Spiricon real-time monitor and compared to the original image to verify correct filtering of the larger array. For further verification, a binary image of a circle and a square were also created inside a 512 x 512 array. A median filter will not affect step changes in intensity. Comparison before and after processing showed no change in either of these two arrays. This routine also calculated the sum of the intensities. This sum was also used as a reference to verify any changes later made in the program.

The next part of the program was the normalization procedure. To verify operation of this routine, scaling factors and random pixel intensities before and after normalization were printed to the screen and compared to hand calculated results. The log compression and the thresholding routine were handled in the same manner.

All routines in *log3x3.c*, including ones not mentioned above, were additionally verified using Norton's HEX editor. Results from *log3x3.c* were used to help verify programs written later.

The *2dft* program was verified by creating 3 files containing a rect function, delta function, and a constant function. The output produced the expected sinc function, constant function, and delta function respectively (2:317). Norton's HEX editor was once again used to further verify results.

Several programs were written in "C" to verify the operation of the Spiricon software. For example, using a program called *oddrow.c* which set odd row pixels to 255 and even row pixels equal to zero, it was discovered that Spiricon only displays every other row of camera data on the real-time monitor. Another program

called *dot.c* set a single pixel to 255 and all others to zero. Spiricon is supposed to access a user created file named BADPIX to ignore this pixel while it processes the data. Through the use of *dot.c*, it was discovered that this feature was not working correctly. See appendix A for further information.

This chapter examined the design of the optical bench. This included a discussion of beam alignment and optical components used. This was followed by remarks on fiber preparation, data gathering, processing, and software verification procedures. The next chapter presents the results of this research and a brief analysis of those results.

IV. Data Analysis

The previous chapter discussed the procedures used to gather and process the data. This chapter analyzes the results obtained from this research. The non symmetry observed in the output pattern for similar angles on opposite sides of boresight is first examined. This is followed by a presentation of the various data gathered and a discussion of the results.

4.1 Discussion of Non-symmetry

During the data gathering portion of this research, a non-symmetry was observed between opposite sides of boresight. This contradicts the belief that the output pattern of a circularly symmetrical fiber should also be symmetrical. This anomaly can be easily explained. When the ends of the fiber are polished, extreme care must be taken to polish the ends orthogonal to the core axis of the fiber. The fibers used in this research were clamped in a fixture fabricated specifically for a *3mm* fiber. The polishing process was completed for each fiber before it was removed from the fixture. Due to the inability to consistently clamp the fiber in the fixture, tolerances in the polishing machine components, such as bearings, and the inability to polish the ends perfectly flat, the entrance and exit of the fiber may neither be parallel to each other nor normal to the core axis. Therefore, when the fiber is mounted in the camera and the core axis is collinear with the laser beam, the beam may contact the entrance to the fiber at an angle other than 0°. The output therefore appears non-symmetrical as the camera is rotated from one side of boresight to the other. This can be demonstrated by rotating the camera while in live mode and watching the output pattern change. Also tolerances in the threads of the C mount fixture and the fact that the hole may have been drilled at a slight angle within the C mount fixture add to the non symmetry. Thus it is important

that tolerances be kept to a minimum with respect to the polishing and mounting of the fiber in a practical application.

4.2 Presentation of Results

Data was gathered for all three lengths of fiber. This consisted of grabbing a frame of data for each degree of incidence from 0° through 25°. Each frame was placed in a file and each file processed to create a 49 component template vector which represented that particular angle of arrival(AOA). All template vectors were placed in the file TEMPLATE.DAT. As stated earlier, the 3*in* fiber was processed first because of the more rapid rate of change of output pattern per degree of change in AOA when compared to the shorter fibers. This was due to the increased number of reflections encountered by the laser light as it propagated through the fiber. Three different processing techniques were used for this fiber. Since the fast Fourier transform(FFT) routine initially required binary input, a threshold of 64(range 0 – 255) was set in the preprocessing program *preproc.c* and the binary option chosen. This threshold was chosen based on comparison of histograms of all template files. Test data was then gathered for various AOA and processed in the same manner. These AOA are shown in column 1 of table 1. The two 7° measurements were taken separately to check repeatability. The Euclidean distance between each test vector and all template vectors was calculated. The minimum distance classified each test vector. The results are shown in the two columns labeled BINARY (64) of table 1. The number in parentheses represents the threshold value chosen. The entry in the distance column represents the distance between the test vector and its closest template vector. It is presented here as a relative comparison rather than absolute measurement. The * in the estimated AOA column indicates correct classification of the test vector. Four test vectors were classified correctly but errors as high as 6 degrees were apparent.

It was believed that some information was lost when the data was binarized so the FFT program was modified to accept non binary data. This non binary

Table 1. Classification of $3in$ Fiber

Distance Between Test and Template Vectors $3in$ Fiber at 0° Reference						
	Binary(64)		Non-Binary		Non-Binary(64)	
Act. AOA	Est. AOA	Distance	Est. AOA	Distance	Est. AOA	Distance
17	*17	29.02	16	12.96	14	28.35
22	20	9.60	21	11.04	20	13.64
24	21	7.88	22	6.78	22	10.49
15	12	68.09	*15	12.96	12	70.01
21	*21	5.48	20	10.00	*21	10.68
3	*3	158.93	*3	44.36	*3	143.35
7	*7	154.40	5	40.80	*7	160.70
7	13	91.86	*7	35.95	13	135.36
11	7	144.35	*11	21.12	7	110.28
	44% correct WCE = 6°		44% correct WCE = 2°		33% correct WCE = 6°	

*Correct Classification
WCE = Worst Case Error

Table 2. Classification of $1in$ and $2in$ Fibers

Distance Between Test and Template Vectors						
	$2in$ Fiber		$1in$ Fiber			
	0° Ref.		0° − 15′ Ref.		0° + 15′ Ref.	
Act. AOA	Est. AOA	Distance	Est. AOA	Distance	Est. AOA	Distance
17	10	23.41	16	27.53	18	26.19
22	17	21.82	*22	22.58	23	17.66
24	*24	16.97	*24	17.55	25	15.49
15	*15	28.71	*15	33.23	*15	35.76
21	22	21.86	*21	30.03	22	21.07
3	*3	101.54	*3	38.65	*3	37.95
7	*7	51.00	6	72.90	*7	56.66
7	*7	46.95	*7	75.17	*7	59.00
11	*11	43.47	*11	43.17	12	38.94
	67% correct WCE = 7°		78% correct WCE = 1°		44% correct WCE = 1°	

*Correct Classification
WCE = Worst Case Error

processing was performed with and without a threshold. The results are shown in the last 4 columns of table 1. The number (64) represents the threshold value. It can be seen that the non-thresholded processing yielded the best results with 41% correct classification and a worst case error(WCE) of only 2°.

The next research effort examined the effect of fiber length on AOA detection. All data was processed non-binary with no threshold, based on the previous results. Test Data was first gathered for the $2in$ fiber. The 0° (boresight) reference was found to be 0° − 15′ on the rotational stage. This was 40′ less than when the template data was taken the month before. This can easily be accounted for. Whenever

the camera is loosened from its mount and allowed to move radially to adjust for different fiber lengths, it may also move very slightly in a lateral direction. Another, more significant, factor concerns the alignment of the laser beam. In the month between when the template data was taken and the test data was taken, the beam steering instrument was accidentally adjusted. This required complete realignment of the beam. In addition, the 0° reference is determined by rotating the camera in live mode until a full intensity circular pattern appears on the screen. This is a subjective measurement and not accurate within minutes of a degree. Taking all these factors into consideration, it was not surprising that boresight had changed from when the template data had been taken. Measurements were taken at the same AOA increments as the $3in$ fiber. Results were favorable for the $2in$ fiber with 67% correct classifications, however, errors were significant with a WCE of 7°. These results are shown in table 2. All angular references in the tables are relative to actual boresight and have no reference to the rotational stage.

4.2.1 Reliability Testing Immediately following this, the camera was realigned and measurements were made on the $1in$ fiber. Three tests were made with the $1in$ fiber to assess reliability in determination of AOA. The first two tests placed the test vectors 15′ on either side of the 0° boresight reference. This was done to verify the ability of the system to continue to correctly determine AOA with a small angular deviation. These results are also presented in table 2. These two tests were successful, both resulting in a WCE of 1°, however the first test performed 34% better, achieving 78% correct classification. The data shows that all errors in the first test were on the low side and all errors in the second test were high. These results were expected since the data was first taken to the negative side of zero and then shifted to the positive side of 0°. These results are also presented in table 2.

4.2.2 Results of Intensity Variation and Non-Symmetry The third reliability test with the $1in$ fiber was combined with two other tests. The first test examined

Table 3. Classification of $1in$ Fiber With Rotation and Variable Intensity

Distance Between Test and Template Vectors $1in$ Fiber at 0° Reference								
	Std. Int.		2 x Std. Int.		.5 x Std. Int.		Rotated 90° Std. Int.	
Act. AOA	Est. AOA	Dist.	Est. AOA	Dist.	Est. AOA	Dist.	Est. AOA	Dist.
17	*17	40.02	0	47.75	16	39.04	3	121.94
22	*22	28.70	*22	55.48	*22	37.87	21	42.33
24	*24	16.12	*24	40.27	25	36.72	22	43.24
15	*15	56.73	*15	80.51	16	37.47	14	101.42
21	*21	36.22	*21	55.46	22	41.16	10	57.43
3	4	60.23	4	104.39	*3	59.24	*3	88.84
7	*7	77.86	*7	91.31	8	59.93	6	146.80
7	*7	75.19	*7	94.48	8	63.07	6	147.08
11	12	44.25	12	61.85	23	46.84	12	96.21
	78% correct WCE = 1°		67% correct WCE = 17°		22% correct WCE = 12°		11% correct WCE = 14°	

*Correct Classification
WCE = Worst Case Error
Std. Int. = Standard Intensity

the affect that a variation in incoming intensity would have on AOA classification. The second measurement involved rotating the fiber 90° to verify that non-symmetry due to the end face being polished non-normal to the core axis would affect correct AOA determination. The benefits of this approach were two-fold. First, this allowed one more check on reliability. More importantly, however, the reliability test provided a reference as to how good the results of the intensity and non-symmetry test really were. The data was gathered in the following manner. Measurements were taken with a 0 degree borsight reference such that there was no angular deviation. The camera was rotated to the first test angle of 17° and the data gathered as before. Then, the .40 OD neutral density filter was removed, more than doubling the intensity(i.e. $10^{.40} = 2.51$), and another frame was grabbed. The .40 OD filter was replaced and another .40 OD filter added, thus decreasing the intensity, and data gathered again. Finally, the added .40 OD filter was removed, the fiber and C-mount threaded fixture rotated 90°, data gathered, and the fixture rotated back to its original setting. This procedure was repeated for each of the nine test angles.

The results are presented in table 3. The first column once again shows actual AOA. The next two columns represent the third reliability test. This test achieved a correct classification of 78% with a WCE of only 1°. Thus the reliability of measurement of AOA using an optical fiber was verified. Using this as a comparison, it can be seen from the other columns in the table that both a change in intensity and the non-orthogonality of the fiber entrance with the core axis affect AOA determination. The adverse effect of intensity in AOA detection is due to normalization based on peak value. This effect could be eliminated through normalization based on total energy. See Chapter 5 for further comments.

The 1*in* fiber appeared to give the best results but this may be misleading. The fibers may have gotten rotated inside their fixtures between template and test vector data gathering. In addition, the measurements performed in this research were subject to human error each time the beam was realigned or the camera ro-

Table 4. Total Intensities and Peak Values for Template Files

Template Intensities Before Normalization						
	3*in* Fiber		2*in* Fiber		1*in* Fiber	
AOA	Total Intensity	Peak	Total Intensity	Peak	Total Intensity	Peak
0	1282365	255	1358228	255	1435971	137
1	1340016	255	1331934	255	1382093	226
2	1490947	255	1365646	255	1359795	224
3	1510842	255	1354307	255	1363492	255
4	1520684	255	1378817	255	1344498	255
5	1507261	255	1357861	255	1368844	255
6	1658921	214	1362819	255	1301121	255
7	1642212	173	1345390	255	1392867	255
8	1608812	89	1393817	255	1300821	255
9	1569449	132	1371979	255	1298955	255
10	1513587	107	1414075	99	1345798	255
11	1515596	177	1320108	107	1278165	255
12	1468427	80	1409195	105	1261141	251
13	1456887	105	1301680	160	1295665	255
14	1439063	65	1294684	214	1247163	217
15	1467542	68	1303516	80	1222484	255
16	1409270	83	1286716	99	1282444	255
17	1492634	70	1280034	110	1199557	255
18	1411670	65	1254365	127	1190632	255
19	1387286	49	1247458	69	1160464	255
20	1425437	59	1226093	63	1171707	226
21	1285504	50	1259666	96	1157413	224
22	1342986	50	1240201	82	1105218	143
23	1130085	39	1216688	96	1124846	98
24	1150165	43	1181791	35	1055813	61
25	1076821	28	1134245	32	1013414	51

Average Noise Intensity = 402,840
Average Peak Noise = 12

tated. The longer fibers may be more sensitive to these slight differences which can occur between tests. It is possible that the longer fibers may outperform the *1in* fiber in more controlled, less subjective conditions. This may also hold true in higher resolution applications where template data is taken in increments of less than 1°. However, a look at table 4 does indicate that the *1in* fiber maintains a substantial peak value for higher AOA's which may play a part in its apparent better performance in AOA classification. Further research will have to be conducted in this area before this can be validated.

4.2.3 Discussion of Thresholding Scheme Looking at table 1, AOA detection was more subject to error when thresholding was set at 64. This number appears to be sufficiently low to eliminate noise while still maintaining enough signal to determine AOA. The reason for the poorer performance can be attributed to two factors. As the AOA increases, more light is reflected off the entrance of the fiber. Thus signal intensity is reduced while noise remains relatively constant. This can be seen in table 4. The first column of each section shows the total intensity and the second column shows the peak value before normalization. Each total intensity value represents the sum of the unsigned bytes(0 — 255) in the reduced 256 x 256 array. This summation was performed in the program *preproc.c*. To give some meaning to these numbers, five reference frames of data were taken with no light on the camera to get some idea of the noise level present. These frames yielded an average intensity of 402,840 and an average peak value of 12 before normalization. For AOA's above 20° the signal-to-noise ratio was low enough to cause problems in AOA classification. Figures 12 and 13 show how the higher peak value patterns at lower AOA's appear before and after normalization as compared to the lower peak value higher AOA's presented in figures 14 and 15. The second factor is program related. The thresholding routine in the program *preproc.c* was originally placed in the program to binarize the data or to select the data above the threshold for viewing or processing. It was not placed in the program to help eliminate noise.

Figure 12. 173 Peak Value and 7° AOA Before Normalization

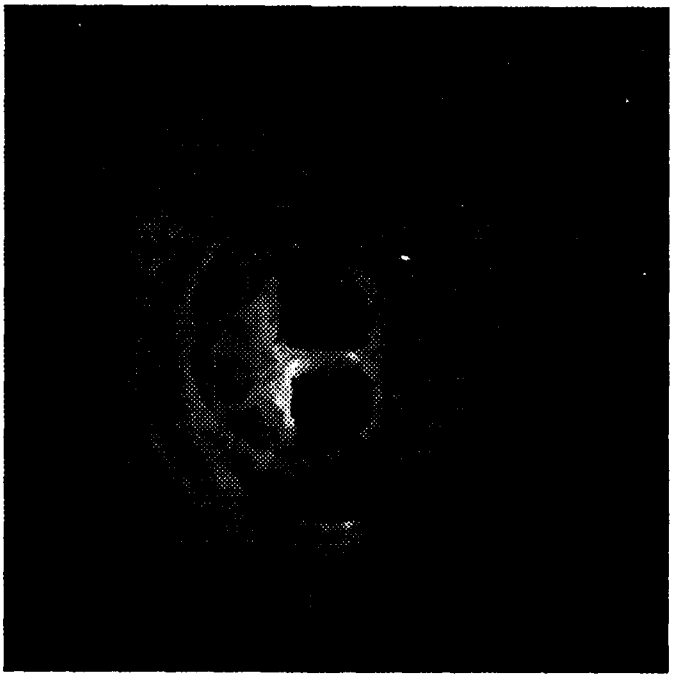


Figure 13. 173 Peak Value and 7° AOA After Normalization

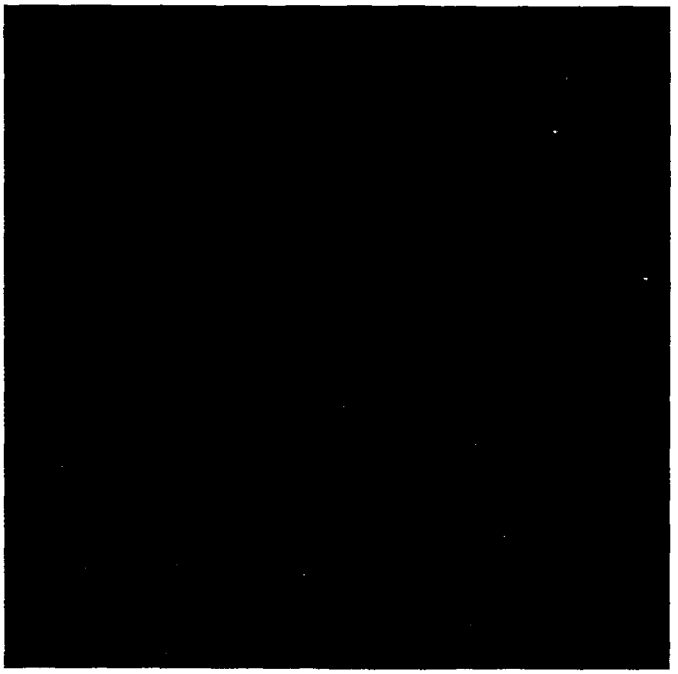


Figure 14. 83 Peak Value and 16° AOA Before Normalization

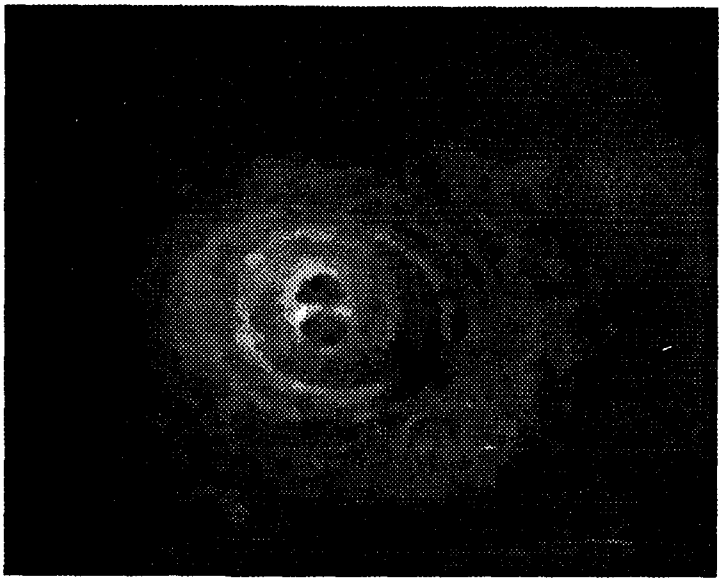


Figure 15. 83 Peak Value and 16° AOA After Normalization

Therefore, it was placed at the end of the program, after the normalization routine. A thresholding routine to reduce noise should be added to the program ahead of the normalization routine. This problem is addressed in the recommendations section of chapter 5.

This chapter presented the data gathered in this research and discussed the results. The next chapter presents conclusions of this research and recommendations for future research in this area.

V. Recommendations For Future Research

5.1 Conclusions

This research was conducted to test the feasibility of using the output patterns from optical fibers to determine angle of arrival(AOA). The experimental optical system was designed and assembled, complete with camera and framegrabber. Parts were fabricated specifically for this research effort to allow for ease and reliability during the data gathering portion of this thesis. These parts included threaded C-mount camera fixtures designed to hold the fibers firmly in place without damage, a fixture mounted between the camera and rotating stage which allowed only radial movement of the camera, and a $3mm$ aperture used to align the end of the fiber with the axis of rotation. Much time was spent learning the 'Turbo C' language and writing the code which performed the signal processing. This was original code written to be compatible with the Spiricon system. This code is presented in the appendices.

Several areas affecting AOA detection were investigated in this thesis. These included various fiber lengths, laser intensities, and processing schemes. The $1in$ *fiber gave the best results* although variations in preprocessing schemes or higher resolution applications may affect this. Variation in source intensity resulted in erroneous AOA classification, demonstrating the need to modify the program *preproc.c* to compensate for this variation. All data was presented as an aid to follow-on research.

A few of the successes achieved in this thesis were presented above, but these are minor compared to the major success of showing that angular resolution of $1°$ or less is possible through the analysis of the output patterns from an optical fiber. Recommendations for future research are presented in the next section.

5.2 *Recommendations*

This section discusses recommendations for future research in AOA detection. Recommendations include upgrades to the optical bench setup, correcting for variable intensity, as well as implementing correlation and neural network processing.

A Newport Equipment Corporation (NEC) model 481 manual rotational stage was used for this research. This rotational stage has a vernier scaled in minutes of degree of arc. The resolution of this stage was adequate for this initial feasibility study, but this stage is small and can support the weight of the CIDTECH camera and cables only if that weight is evenly distributed across the stage. Since the camera and cables are off center from 3 to 6 inches, this stage allows the camera assembly to drop at a slight angle such that it is no longer parallel with the top of the optical bench. Since the laser beam is aligned parallel to the top of the bench, the beam encounters the entrance to the fiber at a slight angle. The use of the NEC model 495 rotational stage is suggested. This stage is larger and has almost three times the load capacity of the model 481. This is also a motorized stage controlled by an IBM or compatible PC using the MS-DOS operating system. This added feature allows stage rotation in increments as small as .001 degree while automatically adjusting for backlash in the mechanism. The NEC 495 should make the job of gathering data less tedious and also allow repeatable performance. This stage was on order at the time this research was performed and should arrive in December 1989.

Limited time was spent on preprocessing schemes. Many approaches can be taken to preprocess the data, each giving different results. Continuing research should include changing the parameters used in the preprocessing in hopes of discovering the best preprocessing scheme. One example would be to analyze the results of log compressing the data since limited time did not allow a chance to invoke this option. In addition, the preprocessing program should be modified to add a noise threshold ahead of the normalization routine.

Processing was made more difficult with the FFT program on the mainframe

and the necessary transfer of files. The files had to be processed one at a time by the user. The program could be modified to run multiple programs without user intervention, but the files would still have to be transferred to the mainframe and back. By completing the work necessary to enable the *fft.c* program to run on the PC, all data gathering and processing can be performed on the same computer. The *fft.c* program can be combined with the *preproc.c*, *template.c*, and *distance.c* programs to process multiple files without user intervention. Add to this the possibility of automatic data gathering with the computer controlled rotating stage triggering the Spiricon frame grabber to grab a frame at each increment of rotation, and further research can be done more efficiently. Once the template files are created the user can move the camera to an arbitrary angle, call a program on the PC which would do all the necessary processing and comparisons to the template file, and find the AOA of the test vector in the output file.

In practice this detector will be subject to various intensities of laser radiation. Further research should involve using the variable attenuator to change the intensity of incoming laser radiation and modifying the preprocessing program to adjust for this change. The normalization program currently in use selects the peak value in each data file and calculates a scaling factor by which to multiply each data point(pixel value) such that the new peak value is 255. This expands the range of values in each file to the maximum but does not adjust the overall intensity of each file to a common reference. Thus the normalization routine should be appended to allow compensation for variable laser intensity by adjusting all files to a common reference before performing the peak normalization.

The only source used in this thesis was a 50mW He-Ne laser with an output wavelength of $632.8nm$. Since Nd-YAG lasers with a wavelength of $1.06\mu m$ are prevalent in military applications for target illumination, the relation of wavelength to the fiber diameter and fiber length should be examined.

Another recommendation involves processing. Currently the processing scheme

involves creating a test feature vector from the low frequency Fourier components and comparing this test vector to previously created template feature vectors using Minkowski distance for classification. This is one of the least effective ways to classify data. In order to utilize information from all frequencies, a correlation between test and template vectors can be performed in the frequency domain. This can be accomplished by modifying existing software such as George LeTourneau's FFT program, using IMSL library programs, or developing a working version of the recently created FFT program written in 'Turbo C' for the PC. To perform the correlation, both the real and imaginary array must be accessible. Take the conjugate of the imaginary component of the template array and then perform a point for point complex multiplication between the template and test arrays. The multiplication which yields the highest correlation peak classifies the test vector. Another processing scheme involves using a Neural Net to classify the test vector. Because Neural Nets are inherently slow, care must be taken in selecting the inputs. One possible set of inputs would be the 49 low frequency Fourier components currently used in the Minkowski distance calculation.

Many data reduction and processing schemes exist and the limited research conducted in this thesis will hopefully provide incentive for further investigation in this area.

Appendix A. *Spiricon 2250 Laser Beam Diagnostics System*

This appendix was written to assist the user in repeating the data collection performed in this thesis. Any information not specifically covered here can be found within one of the two Spiricon manuals retained by AFIT.

The Spiricon 2250 Laser Beam Diagnostics system contains a processing card which uses two slots in an IBM compatible personal computer(PC), version 5.0 Laser Beam Diagnostic software, a CIDTECH 512 x 512 Charge Injection Device(CID) camera, a real-time display monitor, and the necessary connecting cables. The 2250 *system is advertised to work within an* IBM AT, IBM 386, or IBM compatible. After much experimentation with various PC's, it was discovered that the 2250 apparently only works on an IBM AT. This may be due to an addressing problem within the Z-248's and IBM 386 in which the 2250 system failed to operate.

The version 5.0 software is contained on two 360Kbyte floppy disks. The files were copied onto the hard drive into the directory `c:\spir`. Subdirectories were also created to handle the various stages of the data. The subdirectory RAWDATA contained the original frames of data received from the CID camera. These files contain 263,052 unsigned bytes. The structure of these files will be addressed shortly. The subdirectories PROCDATA and FTDATA contained preprocessed data and Fourier transform(FT) data respectively. These files contain 65,536 unsigned bytes. Finally the subdirectories TEMPLATE and TEST contain the feature vectors composed of the low frequency Fourier components representing the shape of the original images. These files vary in length depending on the number of vectors contained in the files. Files containing information relating to the results of data processing are also stored in these last two subdirectories. The files contained in all subdirectories will be examined in greater detail in the upcoming section on processing.

To start Spiricon, type 'Q3D from within the `c:\spir` directory. The main menu will appear on the computer monitor within a few seconds. Several sub-menus

are available from the main menu. Most of these are not of immediate concern and will not be addressed here. Type F7 to open the setup menu. To allow Spiricon to access files in the subdirectories, type [ALT]F2 and the subdirectory name followed by a return. To check operation of the camera, type F2 to access the beamlink menu. Typing F2 again toggles the camera on and off. The camera output is displayed on the real-time monitor. Typing F10 automatically turns the camera off and grabs the frame, as well as returning to the setup menu. Typing F10 again returns to the main menu. Online help is available from within any menu by typing F9.

Now that the camera has been found to be working correctly and the proper subdirectory has been accessed, the functions found within the main and auxiliary menus suffice to carry out the remaining operations needed for this thesis. F1 grabs a new frame of data and displays it on the real-time monitor. Several seconds later a 3-D plot is displayed on the computer monitor. This 3-D plot can be modified in a variety of ways from the auxiliary menu. The 3-D plot is a 128 x 128 representation of the 512 x 512 camera array. The *slice* regulates the resolution of this 128 x 128 array. A slice of 4 provides the highest resolution when looking at the entire(*whole*) 512 x 512 array. A slice of 4 tells Spiricon to represent the average of 4 pixels in the x-direction and 4 pixels in the y-direction as one pixel in the 3-D plot. Slice values can increase by powers of 2 to the upper limit of 64. Large slice values decrease resolution as well as processing time. Pressing M increases the slice value while pressing L will decrease the slice. There are two cursors on the plot represented by crosses at two diagonal corners. In order to achieve maximum resolution(*slice* = 1), the user must zoom in on an area no larger than 128 x 128. This is accomplished by moving the cursors into the plot using the cursor keys on the keyboard. The HOME key toggles between cursors. The coordinates of each cursor are displayed at the bottom of the auxiliary menu. Note that the cursors move in increments based on the slice value. Once the cursors are positioned, press W. This tells Spiricon to plot only the *part* within the cursors. Now press L as necessary to reduce slice to 1. Pressing return

will cause Spiricon to reprocess the image according to the new parameters. There now exists a one to one correspondence between the pixels on the 3-D plot and the pixels on the camera within the representative area. To read the intensity value of a particular pixel, just move the cursor to that pixel and read the value of Z at the bottom of the auxiliary menu. The eight bit intensity range of 0 to 255 is represented in Spiricon by values from 0 to 2040. Another variable is the magnification of the display. If the display is too tall to fit on the screen, the user can reduce the height by pressing - on the keyboard. Typing + as necessary will increase magnification. Values of magnification vary from 1/2 to 8. The display can also be tilted in 5 degree increments from -90 to +90 degrees and rotated in 15 degree increments from -180 to +180 degrees by pressing PgUp/PgDn and Ins/Del respectively. There are ten different formats for the 3-D plot. Some of these add contour lines representing level curves to the plot. These formats are changed each time the X key is pressed. Once the desired parameters are set, they can be saved by pressing F7 to access the setup menu. Now press [Alt]F1 to save the configuration. Press F10 to return to the main menu. The parameters are saved in the file SPIR.FIG which is 906 bytes long. Whenever Spiricon is initially loaded, it looks for the file SPIR.FIG and sets the parameters accordingly. There are two ways to send the plot to a printer. Typing P performs a screen dump. Thus menus are printed as will as the plot. To print the 3-D plot and various other information without the menus, press F2. To save the plot to disk with the current parameters, press F4, enter a filename of your choice, and press return. The user is now prompted for a record number. Spiricon allows as many as 32,000 records to be stored under each filename. This is overkill since each record is 263,053 bytes long. Type 1 and hit return. If record 1 already exists, the user can overwrite it or choose another record or filename. The plot is stored with the parameters currently in use. Once the file is stored according to the data path selected earlier, the main menu once again appears on the screen. To load another file, press F3. The user is given a list of files in the directory chosen earlier. Press

the up or down cursor keys to select a file and hit return. If the required file is in another directory the user can change directories by pressing D and typing the new directory path. This is an alternate way of changing directory path. Files can also be deleted by selecting a file with the cursor keys and pressing the DEL key. Once a file is selected for loading, type return. The chosen file is loaded, displayed on the real-time monitor, sliced, and then plotted on the computer monitor according to the current parameters. Sometimes it is desired to display the file with the parameters set as they were when the file was stored. This is achieved by typing F7 from the main menu to access the setup menu before trying to load the file. Type F7 again to access the miscellaneous menu. Now type F6. This tells Spiricon to load files and set parameters according to the information stored in the files. Type F10 twice to return to the main menu. To quit this feature, repeat the procedure.

Two anomalies were noted while using the Spiricon system. It was noted that Spiricon gave an error message when trying to load a file from floppy disk although this is an allowed function. Thus any data files from another system converted to Spiricon format had to be copied to the hard drive before being loaded. The second anomaly concerns a user created file called BADPIX. This file contains coordinates of bad pixels located on the camera sensor. These bad pixels are very sensitive and register at the maximum value when little or no light is present. This resulted in the warning message "Usable range of camera exceeded". This message occurs whenever intensity values exceed approximately 80% of peak. Spiricon is supposed to examine this file and ignore these bad pixels in the data display. This file was never used successfully although calls were made to Spiricon Incorporated and an updated version of software was finally sent.

As mentioned earlier, Spiricon stores the data received from the camera in files 263,052 bytes in length. The first 2 bytes contain the HEX digits 39 and 30. These are used by Spiricon to find the beginning of a data file. The next 262,144 bytes are the data from the camera. The last 906 bytes contain the display parameters which were

in effect when the file was stored. Notice this is equivalent to the SPIR.FIG file. The data is written to the real-time monitor starting at the top and working downward, however, only every other row of data supplied by the camera is displayed on the real-time monitor. All columns are displayed. Although the first row of data(512 bytes) in the file represents the top row of the real-time monitor, it is listed as row 512 on the 3-D plot since the origin is in the lower left hand corner of the image. All bytes in the last 11 rows in the data file, corresponding to the first 11 rows on the 3-D plot, are always zero.

Spiricon has many more features such as gain adjustment, curvefit, and capture length that are not addressed here. Spiricon can also be used for continuous or pulsed lasers.

Appendix B. *Program LOG3X3.C*

The program *log3x3.c* is used on full size Spiricon files. This program does not have any reduction routines, therefore, the output remains a full size Spiricon file. The program is written in loops. There is one large outside loop containing five subroutines. These subroutines are median filter, normalization, thresholding, peak value, and random access. The last two subroutines are contained in individual loops to allow repetition without running through the entire program each time. The flow chart in figure 16 demonstrates this process.

Upon entering the program the user is asked if the data is to be median filtered. If the user answers yes then he is prompted for a data path and filename without the three letter extension. The filename in this case cannot have an extension. The program then reads FILENAME and the data is median filtered while the peak intensity and the sum of the intensities is also calculated. The median filtered data is automatically placed in FILENAME.MD3 while the intensity data is displayed on the monitor. The program now asks the user if he wishes to normalize to the peak value. If yes, then the program automatically reads FILENAME.MD3 and opens output file FILENAME.NM3. The user can choose the value he wishes to set the peak value (i.e. normalize) to, log compress the data if so desired, and also display a histogram of the data on the monitor. For example, if the peak value of the data is 119 and the user wishes to normalize to 255, then a scaling value is calculated which converts 119 to 255. The scaling value is then multiplied by each data point in the array. Log compression calculates the logarithm of each data point in base 10, calculates a different scaling factor, and then normalizes. This option effectively raises the lower intensities such that the data is not spread over a large range of values. The normalization routine obtains the peak value from the median filter routine. Once the data is normalized, the data is segmented into 16 groups of 16

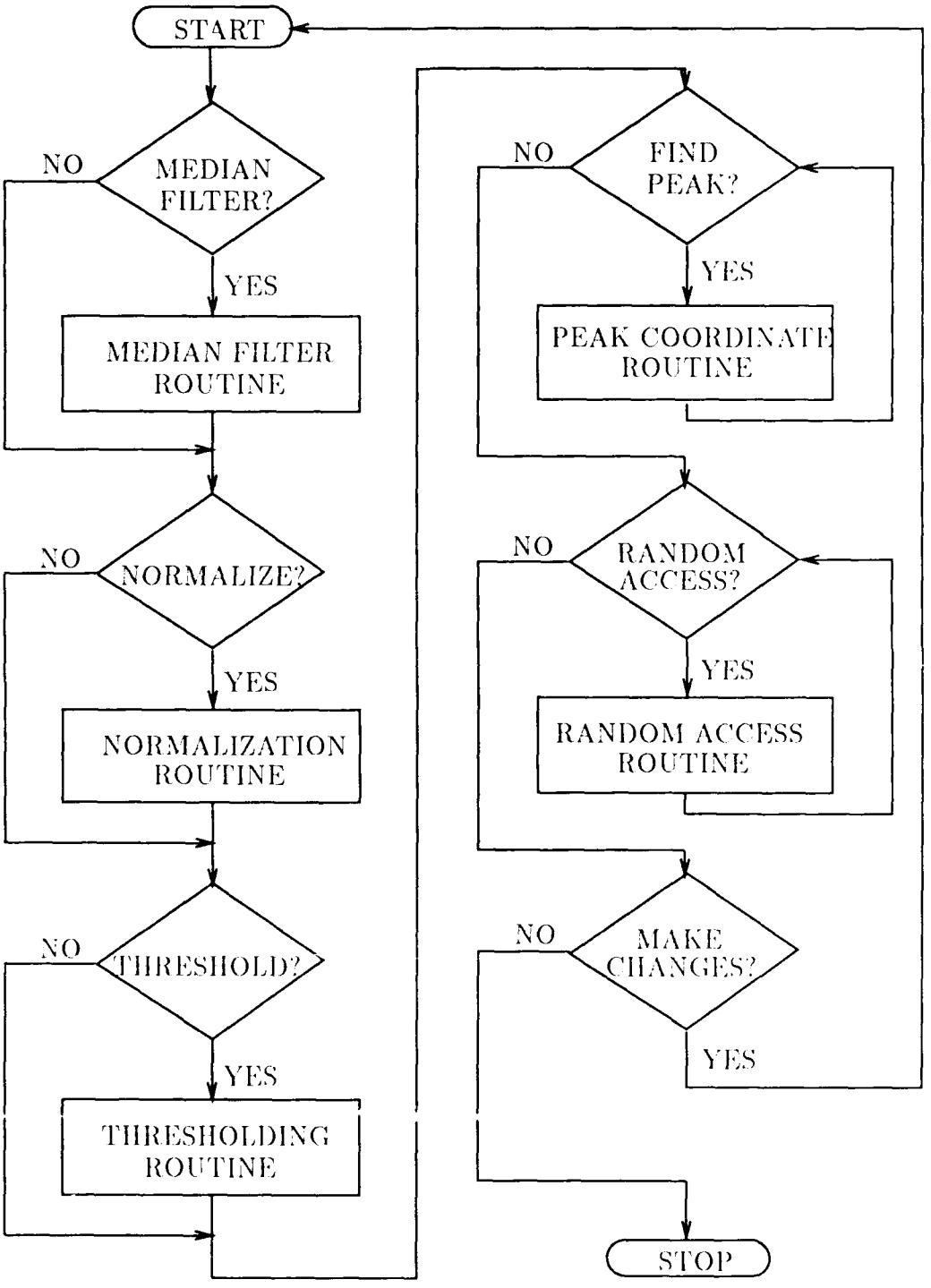


Figure 16. Flowchart For Program *log3x3.c*

values. This information is displayed on the monitor upon entering the thresholding routine and is used to help choose a threshold value.

If the data was not median filtered, the program, upon entering the normalization routine, will ask for an input filename without the three letter extension . This time, however, the program appends the .MD3 extension to the filename which was entered, such that the normalization routine still reads FILENAME.MD3. It then calculates the peak value since it was not calculated in the median filter routine. The normalization routine then runs as previously described.

Following the normalization routine is the thresholding routine. It automatically reads FILENAME.NM3 and opens output file FILENAME.TH3. If the histogram option was chosen in the normalization routine, it is displayed on the screen at this time. An example of this display is shown in figure 11 in chapter 3. The program then displays the peak value of the data file and asks the user to enter a threshold value. All values below threshold are set to zero. The user can also decide to binarize the data. If this option is chosen, then all the values above the threshold are set to 255. Otherwise the values above the threshold remain as they are.

If the normalization routine was not entered, then the program asks for the directory path and filename without extension. It automatically appends the extension .NM3, such that it still reads FILENAME.NM3. The program then asks if a histogram of the data is desired. The program then calculates the maximum intensity and segments the data. It then displays a histogram if that option was chosen. Following this the program performs as described in the previous paragraph.

The next routine is dedicated to finding the coordinates of the peak intensity, value of the peak intensity, and the sum of the intensities. This routine expects a data path and filename with extension. Thus, it does not require a specific extension since the user enters this information. Since this program lies within a while loop, it can be re entered immediately upon completion and given a new filename to access.

Following this routine is a random access routine. This allows the user to display a particular area of data within a file on the monitor. The user once again chooses a data path and filename with extension. The user also chooses the beginning row and column and the ending row and column. There are 512 columns per row in the data file, however, the display on the screen is only 16 columns wide. Thus, one row from the data file requires 16 rows on the screen. This routine also lies within a while loop and can be accessed as many times as necessary. Upon completion of this routine, the program queries the user for changes. If the user wishes to make changes, the program returns to the beginning and starts over. If changes are not required, the program is terminated.

It should be noted that the routines display the data pointer location on the screen while the program is running. In some cases the pointer may indicate that the program did not access all 262,144 bytes of data. This is because Spiricon places zeros in the last 5632 bytes of the file. Thus, some of the calculations, such as the intensity summation, do not need to operate beyond the first 501 rows of the file.

In summary, the median filter routine reads a filename with no extension while the normalization routine reads a filename with the .MD3 extension and the thresholding routine reads a filename with the .TH3 extension, even though the user only enters a data path and filename with no extension (i.e. `c:\spir\rawdata\myfile`). The program was designed in this manner so that a user could go smoothly through each of the first three routines, exit the program, and compare the results of each routine since each routine sends it's output data to a different file.

```
/*Program LOG3X3.C*/

/*Written by Capt John W. Welker*/



/*This program performs the pre-processing on a Spiricon image file*/
/*This pre-processing consists of a median filter operation which*/
/*removes spikes in the data. The data is then energy normalized on*/
/*a linear scale. The data may be log compressed to increase the*/
/*lower intensities. A thresholding option is also available which*/
/*will set all bits below the threshold to zero. Bits above*/
/*threshold can be set to 255 or left unchanged. An option to find*/
/*the coordinates of the peak value and to display a random area of*/
/*data is also available. Recursion means that the median value is*/
/*placed into the middle of the 3 x 3 matrix before it calculates*/
/*the next median value.*/

/*COMPILER DIRECTIVES*/
#include <stdio.h>
#include <string.h>
#include <math.h>

/*DEFINE MACROS*/
#define inrow 104
#define incol 512
#define slice 5

/*DECLARE POINTERS TO FILE STRUCTURE*/
FILE *ptr1,*ptr2;

/*START MAIN PROGRAM*/
void main()
{

/*DECLARE LOCAL VARIABLES*/
unsigned char filename[256],median_file[256],array[inrow][incol];
unsigned char temp[9],random_file_access[256];
unsigned char norm_file[256],thresh_file[256],max_file[256];
unsigned record_num,row,col,max=0,count1,count2,count3,c,r,a=0;
```

```
unsigned length=0,peak=0,norm_max,threshold=0;
unsigned begin_row,begin_col,end_row,end_col,random_length=0;
undigned random_row=0,random_col=0,peakmax=0,peakrow=0,peakcol=0;
unsigned long sum=0,offset=0,num_byte=0,slice_sum;
unsigned long norm_sum,ptr_offset=0,peakbyte=0;
unsigned long segment1,segment2,segment3,segment4,segment5,segment6;
unsigned long segment7,segment8,segment9,segment10,segment11;
unsigned long segment12,segmenti3,segment14,segment15,segment16;
char program='y',median,normal,thresh,random,binary,maxbyte;
char log_scale,histo,header[] = {'\x39', '\x30'};
float norm_scale=0.0,thresh_scale=0.0;


length=inrow*incol;


while(program=='y')
{
printf("\n\nDo you wish to smooth data with a median filter,
y or n? \n");
scanf(" %c",&median);
if(median=='y')
{
sum=0;
max=0;
num_byte=0;
peakmax=0;
peakrow=0;
peakcol=0;
peakbyte=0;

/*ENTER FILES AND VARIABLE VALUES FROM KEYBOARD*/
printf("\nEnter directory path and file name of input file without
extension: \n\n");
scanf("%s",filename);
printf("\nEnter record number \n");
scanf("%u",&record_num);

sprintf(median_file,"%s.md3",filename);

/*OPEN INPUT AND OUTPUT FILES*/
printf("\nOpening input file %s \n",filename);
```

```
if( (ptr1=fopen(filename,"rb"))==NULL)
    {
    printf("Can't open input file %s \n",filename);
    exit();
    }
printf("Opening output file %s \n\n",median_file);
if( (ptr2=fopen(median_file,"wb"))==NULL)
    {
    printf("Can't open output file %s \n",median_file);
    exit();
    }

/*MOVE POINTER TO DESIRED RECORD IN FILE*/
offset=(record_num-1)*263052+2;
fseek(ptr1,offset,0);

fwrite(header,1,2,ptr2);


                    /*BEGIN MEDIAN FILTER PROCESSING*/

for(count1=0;count1<slice;count1++)
    {
    if(count1==0)                                   /*FIRST PASS ONLY*/
{
printf("reading slice #%u \n",count1+1);
fread(array,1,length,ptr1);         /*FIRST READ INTO ARRAY*/
printf("ptr1 = %lu \n\n",ftell(ptr1));
r=0;
for(count2=0;count2<inrow-2;count2++)     /*FIRST MEDIAN*/
   /*FILTER LOOP*/
    {
    c=0;
    for(count3=0;count3<incol-2;count3++)
{
a=0;
for(row=r;row<r+3;row++)        /*SELECT VALUES FOR*/
    {                               /*3 X 3 ARRAY*/
    for(col=c;col<c+3;col++) /*AND PREPARE FOR SORT*/
{
temp[a]=array[row][col];
a++;
```

```
}
    }
sort(temp,a);                    /*SORT  3 X 3 ARRAY*/
c++;
array[r+1][c]=temp[a/2];    /*PLACE MEDIAN VALUE IN*/
}                           /*CENTER OF 3 X 3 ARRAY*/
    r++;
    }
for(col=0;col<incol;col++)          /*SET FIRST ROW EQUAL*/
    array[0][col]=array[1][col];       /*TO SECOND ROW*/
for(row=0;row<inrow-2;row++)
    {                                /*RESET FIRST AND LAST*/
    array[row][0]=array[row][1];           /*COLUMN*/
    array[row][incol-1]=array[row][incol-2];
    }
printf("Calculating first slice intensity and maximum \n\n");
slice_sum=0;
for(row=0;row<inrow-2;row++)      /*CALCULATE MAXIMUM VALUE*/
    {                                  /*AND INTENSITY*/
    for(col=0;col<incol;col++)
{
sum+=array[row][col];
slice_sum+=array[row][col];
num_byte++;
max = (max > array[row][col]) ? max :array[row][col];
if(peakmax != max)
    {
    peakmax=max;
    peakbyte=num_byte;
    }
}
    }
printf("The sum of this slice is %lu \n\n",slice_sum);
printf("Writing slice #%u \n",count1+1);
       fwrite(array,1,length-2*incol,ptr2);/*WRITE ALL BUT LAST TWO*/
printf("ptr2 = %lu \n\n",ftell(ptr2));/*ROWS TO OUTPUT FILE*/
}
    else                                    /*BEGIN ALL SUCCESSIVE*/
{                                  /*PASSES THROUGH FILTER*/
for(row=0;row<2;row++)
    {
    for(col=0;col<incol;col++)   /*LAST TWO ROWS ARE NOW*/
```

```
{                                /*FIRST TWO ROWS*/
        array[row][col]=array[inrow-2+row][col];
}
    }
printf("reading slice #%u \n",count1+1);
fread(&array[2][0],1,length-2*incol,ptr1);
       /*READ IN SECOND SLICE OF*/
/*DATA STARTING IN THIRD*/
    /*ROW OF ARRAY*/

printf("ptr1 = %lu \n\n",ftell(ptr1));
r=0;
for(count2=0;count2<inrow-2;count2++)
    {                              /*SUCCESSIVE MEDIAN FILTER*/
    c=0;                                        /*LOOPS*/
    for(count3=0;count3<incol-2;count3++)
{
a=0;
for(row=r;row<r+3;row++)  /*SELECT VALUES FOR 3 X 3*/
    {                       /*ARRAY AND PREPARE FOR*/
    for(col=c;col<c+3;col++)       /*SORTING*/
{
temp[a]=array[row][col];
a++;
}
    }
sort(temp,a);                  /*SORT 3 X 3 ARRAY*/
c++;
array[r+1][c]=temp[a/2];  /*PLACE MEDIAN VALUE INTO*/
}                           /*CENTER OF 3 X 3 ARRAY*/
    r++;
    }
for(row=0;row<inrow-2;row++)        /*RESET FIRST AND LAST*/
    {                                       /*COLUMNS*/
    array[row][0]=array[row][1];
    array[row][incol-1]=array[row][incol-2];
    }
if(count1<slice-1)                    /*INTERMEDIATE LOOPS*/
            {
    printf("Calc lating intermediate slice intensity and
    maximum \n\n");
    slice_sum=0;
```

```
    for(row=0;row<inrow-2;row++)
{
for(col=0;col<incol;col++)/*CALCULATE MAXIMUM VALUE*/
    {                          /*AND INTENSITY*/
    sum+=array[row][col];
    slice_sum+=array[row][col];
    num_byte++;
    max =
    (max > array[row][col]) ? max : array[row][col];
    if(peakmax != max)
     {
     peakmax=max;
     peakbyte=num_byte;
     }
    }
}
    printf("The sum of this slice is %lu \n\n",slice_sum);
    printf("Writing slice #%u \n",count1+1);
    fwrite(array,1,length-2*incol,ptr2);
    /*WRITE TO OUTPUT FILE*/
    /*DURING INTERMEDIATE*/
    /*PASSES THROUGH FILTER*/
    printf("ptr2 = %lu \n\n",ftell(ptr2));
    }
else
    {
      /*LAST PASS THROUGH FILTER*/
    /*RESET FIRST AND LAST COLUMN*/
    array[inrow-2][0]=array[inrow-2][1];
    array[inrow-2][incol-1]=array[inrow-2][incol-2];
    for(col=0;col<incol;col++)             /*RESET LAST ROW*/
array[inrow-1][col]=array[inrow-2][col];
    printf("Calculating last intensity and maximum \n\n");
    slice_sum=0;
    for(row=0;row<inrow;row++)    /*CALCULATE MAXIMUM VALUE*/
{                          /*AND INTENSITY IN LAST*/
for(col=0;col<incol;col++)         /*SLICE*/
    {
    sum+=array[row][col];
    slice_sum+=array[row][col];
    num_byte++;
    max =
```

```
    (max > array[row][col]) ? max : array[row][col];
    if(peakmax != max)
     {
     peakmax=max;
     peakbyte=num_byte;
     }
    }
}
    printf("The sum of this slice is %lu \n\n",slice_sum);
    printf("Writing slice #%u \n",count1+1);
    fwrite(array,1,length,ptr2);  /*WRITE ALL OF LAST SLICE*/
    printf("ptr2 = %lu \n\n",ftell(ptr2)); /*TO OUTPUT FILE*/
    }
}
    }
if(length>906)
    {
    fread(array,1,906,ptr1);
    fwrite(array,1,906,ptr2);
    }
fclose(ptr1);                                    /*CLOSE FILES*/
fclose(ptr2);
peakrow=(peakbyte/(long)incol);
peakcol=(peakbyte % (long)incol)-1;
printf("The sum of the first %lu intensities is %lu \n\n",
num_byte,sum);
printf("The peak intensity of %u after filtering occurs at \n",max);
printf("row %u and column %u in file %s. \n\n",peakrow,peakcol,
median_file);
printf("This corresponds to coordinates (%u,%u) ",peakcol,
512-peakrow);
printf("on Spiricon plot. \n");
}


                    /*BEGINNING NORMALIZATION ROUTINE*/

printf("\n\nDo you wish to normalize, y or n? \n");
scanf(" %c",&normal);
if(normal=='y')
    {
    norm_sum=0;
```

```
    norm_max=0;
    num_byte=0;
    segment1=0;
    segment2=0;
    segment3=0;
    segment4=0;
    segment5=0;
    segment6=0;
    segment7=0;
    segment8=0;
    segment9=0;
    segment10=0;
    segment11=0;
    segment12=0;
    segment13=0;
    segment14=0;
    segment15=0;
    segment16=0;
    if(median!='y')
{
max=0;
printf("\nEnter directory path and file name of input file
without extension: \n\n");
scanf("%s",filename);
        sprintf(median_file,"%s.md3",filename);
        printf("\nOpening input file %s \n\n",median_file);
if( (ptr1=fopen(median_file,"rb"))==NULL)
    {
    printf("Can't open input file %s \n",median_file);
    exit();
    }
fseek(ptr1,2,0);
printf("\nCalculating maximum intensity \n\n");
for(count1=0;count1<slice;count1++)
    {
    printf("reading slice #%u \n",count1+1);
    fread(array,1,length-2*incol,ptr1);
    printf("ptr1=%lu \n\n",ftell(ptr1));
    for(row=0;row<inrow-2;row++)
{
for(col=0;col<incol;col++)
    {
```

```
    max=(max>array[row][col]) ? max :array[row][col];
    }
}
    }
printf("The maximum intensity in file %s is %u \n",
median_file,max);
fclose(ptr1);
}
    sprintf(norm_file,"%s.nm3",filename);
    printf("\nWhat number do you wish to normalize to (2040 is
    peak)? \n");
    scanf("%u",&peak);
    printf("\nHighest intensity will be %u \n",peak);
    printf("Do you wish to perform a log compression of the data,
    y or n? \n");
    scanf(" %c",&log_scale);
    if(log_scale=='y')
norm_scale=(float)peak*255.0/2040.0/log((double)max);
    else
norm_scale=((float)peak+0.5)*255.0/2040.0/(float)max;
    printf("Scaling factor is %f \n\n",norm_scale);
    printf("Do you wish to see a histogram of the data ");
    printf("for thresholding purposes, y or n? \n");
    scanf(" %c",&histo);
    printf("\nOpening input file %s for normalization \n",
    median_file);
    ptr1=fopen(median_file,"rb");
    printf("Opening output file %s for normalization\n\n",norm_file);
    ptr2=fopen(norm_file,"wb");
    fseek(ptr1,2,0);
    fwrite(header,1,2,ptr2);
    for(count1=0;count1<slice;count1++)
{
printf("reading slice #%u \n",count1+1);
fread(array,1,length-2*incol,ptr1);
printf("ptr1=%lu \n\n",ftell(ptr1));
printf("Value of array[1][1] before scaling is %u \n",
array[1][1]);
for(row=0;row<inrow-2;row++)
    {
    for(col=0;col<incol;col++)
{
```

```
if(log_scale=='y')
    {
    array[row][col]=log((double)array[row][col]+1);
    if(row==1 && col==1)
printf("Value of array[1][1] after log
compression is %u \n",array[1][1]);
    }
array[row][col]*=norm_scale;
norm_sum+=array[row][col];
norm_max=
(norm_max>array[row][col]) ?norm_max:array[row][col];
if(histo=='y')
                    {
    if(array[row][col]<16)
segment1++;
    else if(array[row][col]<32)
segment2++;
    else if(array[row][col]<48)
segment3++;
    else if(array[row][col]<64)
segment4++;
    else if(array[row][col]<80)
segment5++;
    else if(array[row][col]<96)
segment6++;
    else if(array[row][col]<112)
segment7++;
    else if(array[row][col]<128)
segment8++;
    else if(array[row][col]<144)
segment9++;
    else if(array[row][col]<160)
segment10++;
    else if(array[row][col]<176)
segment11++;
    else if(array[row][col]<192)
segment12++;
    else if(array[row][col]<208)
segment13++;
    else if(array[row][col]<224)
segment14++;
    else if(array[row][col]<240)
```

```
segment15++;
    else if(array[row][col]<=255)
segment16++;
    num_byte++;
    }
}
    }
printf("Value of array[i][1] after scaling is %u \n\n",
array[1][1]);
printf("Writing slice #%u \n",count1+1);
fwrite(array,1,length-2*incol,ptr2);
printf("ptr2=%lu \n\n",ftell(ptr2));
}
    printf("The sum of the intensities after normalization
    is %lu \n",norm_sum);
    printf("The normalized maximum is %u \n",norm_max);
    fread(array,1,1930,ptr1);
    fwrite(array,1,1930,ptr2);
    fclose(ptr1);
    fclose(ptr2);
    }

                    /*BEGINNING THRESHOLDING ROUTINE*/

printf("\n\nDo you wish to set a threshold, y or n? \n");
scanf(" %c",&thresh);
if(thresh=='y')
    {
    if(normal!='y')
{
max=0;
num_byte=0;
segment1=0;
segment2=0;
segment3=0;
segment4=0;
segment5=0;
segment6=0;
segment7=0;
segment8=0;
segment9=0;
segment10=0;
```

```
segment11=0;
segment12=0;
segment13=0;
segment14=0;
segment15=0;
segment16=0;
printf("\nEnter directory path and file name of input file
without extension: \n\n");
scanf("%s",filename);
        sprintf(norm_file,"%s.nm3",filename);
        printf("\nOpening input file %s \n\n",norm_file);
if( (ptr1=fopen(norm_file,"rb"))==NULL)
    {
    printf("Can't open input file %s \n",norm_file);
    exit();
    }
fseek(ptr1,2,0);
printf("\nDo you wish to see a histogram of the data,
y or n? \n");
     scanf(" %c",&histo);
printf("\nCalculating maximum intensity \n\n");
for(count1=0;count1<slice;count1++)
    {
    printf("reading slice #%u \n",count1+1);
    fread(array,1,length-2*incol,ptr1);
    printf("ptr1=%lu \n\n",ftell(ptr1));
    for(row=0;row<inrow-2;row++)
{
for(col=0;col<incol;col++)
    {
    max=(max>array[row][col]) ? max :array[row][col];
    if(histo=='y')
                        {
if(array[row][col]<16)
    segment1++;
else if(array[row][col]<32)
    segment2++;
else if(array[row][col]<48)
    segment3++;
else if(array[row][col]<64)
    segment4++;
else if(array[row][col]<80)
```

```
    segment5++;
else if(array[row][col]<96)
    segment6++;
else if(array[row][col]<112)
    segment7++;
else if(array[row][col]<128)
    segment8++;
else if(array[row][col]<144)
    segment9++;
else if(array[row][col]<160)
    segment10++;
else if(array[row][col]<176)
    segment11++;
else if(array[row][col]<192)
    segment12++;
else if(array[row][col]<208)
    segment13++;
else if(array[row][col]<224)
    segment14++;
else if(array[row][col]<240)
    segment15++;
else if(array[row][col]<=255)
    segment16++;
num_byte++;
}
    }
}
    }
printf("The maximum intensity in file %s is %u \n",
norm_file,max);
peak=max*2040.0/255.0;
fclose(ptr1);
}
    if(histo=='y')
{
printf("\n\n\n\t\t\t\tHISTOGRAM OF DATA \n");
printf("\t\t(Value in parentheses is value on spiricon
plot) \n\n\n");
printf("%.2f \tpercent of the values lie between
0(0)         and    15(127). \n",(float)segment1/
(float)num_byte*100.0);
printf("%.2f \tpercent of the values lie between
```

```
16(128)      and    31(255). \n",(float)segment2/
(float)num_byte*100.0);
printf("%.2f \tpercent of the values lie between
32(256)      and    47(382). \n",(float)segment3/
(float)num_byte*100.0);
printf("%.2f \tpercent of the values lie between
48(383)      and    63(510). \n",(float)segment4/
(float)num_byte*100.0);
printf("%.2f \tpercent of the values lie between
64(511)      and    79(637). \n",(float)segment5/
(float)num_byte*100.0);
printf("%.2f \tpercent of the values lie between
80(638)      and    95(765). \n",(float)segment6/
(float)num_byte*100.0);
printf("%.2f \tpercent of the values lie between
96(766)      and    111(892). \n",(float)segment7/
(float)num_byte*100.0);
printf("%.2f \tpercent of the values lie between
112(893)     and    127(1020). \n",(float)segment8/
(float)num_byte*100.0);
printf("%.2f \tpercent of the values lie between
128(1021)    and    143(1147). \n",(float)segment9/
(float)num_byte*100.0);
printf("%.2f \tpercent of the values lie between
144(1148)    and    159(1275). \n",(float)segment10/
(float)num_byte*100.0);
printf("%.2f \tpercent of the values lie between
160(1276)    and    175(1402). \n",(float)segment11/
(float)num_byte*100.0);
printf("%.2f \tpercent of the values lie between
176(1403)    and    191(1530). \n",(float)segment12/
(float)num_byte*100.0);
printf("%.2f \tpercent of the values lie between
192(1531)    and    207(1657). \n",(float)segment13/
(float)num_byte*100.0);
printf("%.2f \tpercent of the values lie between
208(1658)    and    223(1785). \n",(float)segment14/
(float)num_byte*100.0);
printf("%.2f \tpercent of the values lie between
224(1786)    and    239(1912). \n",(float)segment15/
(float)num_byte*100.0);
printf("%.2f \tpercent of the values lie between
```

```
240(1913)    and    255(2040). \n",(float)segment16/
(float)num_byte*100.0);
}
    printf("\n\nEnter threshold value (%u is peak)? \n",peak);
    scanf("%u",&threshold);
    printf("\nDo you wish to set intensities above the threshold ");
    printf("to a single value, y or n? \n");
    scanf(" %c",&binary);
    if(binary=='y')
{
     printf("\nEverything below %u will be zero ",threshold);
printf("and everything above %u will equal 2040 \n",
threshold);
}
    else
     printf("\nEverything below %u will be zero \n",threshold);
    thresh_scale=(float)threshold*255.0/2040.0;
    printf("Threshold equals %f \n\n",thresh_scale);
    sprintf(thresh_file,"%s.th3",filename);
    printf("Opening input file %s for thresholding \n",norm_file);
    ptr1=fopen(norm_file,"rb");
    printf("Opening output file %s for thresholding \n\n",
    thresh_file);
    ptr2=fopen(thresh_file,"wb");
    fseek(ptr1,2,0);
    fwrite(header,1,2,ptr2);
    for(count1=0;count1<slice;count1++)
{
printf("reading slice #%u \n",count1+1);
fread(array,1,length-2*incol,ptr1);
printf("ptr1=%lu \n\n",ftell(ptr1));
for(row=0;row<inrow-2;row++)
    {
    for(col=0;col<incol;col++)
{
                if(binary=='y')
    {
    if(array[row][col]<thresh_scale)
        array[row][col]=0;
    else
array[row][col]=255;
                    }
```

```
else
    {
    if(array[row][col]<thresh_scale)
        array[row][col]=0;
    }
}
    }
printf("Writing slice #%u \n",count1+1);
fwrite(array,1,length-2*incol,ptr2);
printf("ptr2=%lu \n\n",ftell(ptr2));
}
    fread(array,1,1930,ptr1);
    fwrite(array,1,1930,ptr2);
    fclose(ptr1);
    fclose(ptr2);
    }

                /*BEGINNING MAXBYTE AND SUMMATION ROUTINE*/

printf("\n\nDo you wish to find the coordinates of the peak
intensity, y or n? \n");
scanf(" %c",&maxbyte);
while(maxbyte=='y')
    {
    max=0;
    sum=0;
    num_byte=0;
    printf("\nEnter directory path and file name of input file
    : \n\n");
    scanf("%s",max_file);
    printf("\nEnter record number \n");
    scanf("%u",&record_num);
    printf("\nOpening input file %s \n",max_file);
    if( (ptr1=fopen(max_file,"rb"))==NULL)
{
printf("\nCan't open input file %s \n",max_file);
exit();
}

/*MOVE POINTER TO DESIRED RECORD IN FILE*/
    offset=(record_num-1)*263052+2;
    fseek(ptr1,offset,0);
```

```
    printf("\nCalculating maximum intensity \n\n");
    for(count1=0;count1<slice;count1++)
{
printf("reading slice #%u \n",count1+1);
fread(array,1,length-2*incol,ptr1);
printf("ptr1=%lu \n\n",ftell(ptr1));
for(row=0;row<inrow-2;row++)
    {
    for(col=0;col<incol;col++)
{
sum+=array[row][col];
num_byte++;
max=(max>array[row][col]) ? max : array[row][col];
if(peakmax != max)
    {
    peakmax=max;
    peakbyte=num_byte;
    }
}
    }
}
    fclose(ptr1);
    peak=max*2040.0/255.0;
    peakrow=(peakbyte/(long)incol);
    peakcol=(peakbyte % (long)incol)-1;
    printf("The sum of the first %lu intensities is %lu \n\n",
    num_byte,sum);
    printf("The peak intensity of %u occurs at row %u and \n",max,
    peakrow);
    printf("column %u in file %s. \n\n\n",peakcol,max_file);
    printf("This corresponds to a peak of %u at ",peak);
    printf("coordinates (%u,%u) on Spiricon plot. \n\n",peakcol,
    512-peakrow);
    printf("Do you wish to find the peak and sum in another
    file, y or n? \n");
    scanf(" %c",&maxbyte);
    }

                    /*BEGINNING RANDOM ACCESS ROUTINE*/

printf("\n\nDo you wish to look at a particular area of
data, y or n? \n");
```

```
scanf(" %c",&random);
while(random=='y')
    {
    printf("\nEnter beginning row and column starting at 0,0
    (i.e.   24,105   )\n");
    scanf("%u,%u",&begin_row,&begin_col);
    printf("The beginning row and column is %u,%u \n\n",begin_row,
    begin_col);
    printf("Enter ending row and column \n");
    scanf("%u,%u",&end_row,&end_col);
    printf("The ending row and column is %u,%u\n\n",end_row,end_col);
    ptr_offset=(long)begin_row*(long)incol+begin_col+2;
    random_length=(end_row*incol+end_col+2)-ptr_offset;
    random_row=random_length/incol;
    random_col=random_length%incol;
    printf("The pointer offset = %lu \n",ptr_offset);
    printf("The random length = %u \n\n",random_length);
    if(random_length<length)
{
printf("Enter directory path and file name of input file
: \n\n");
     scanf("%s",random_file_access);
printf("\nOpening input file %s for random access \n\n",
random_file_access);
if( (ptr1=fopen(random_file_access,"rb"))==NULL)
    {
    printf("Can't open input file %s \n",random_file_access);
    exit();
    }
printf("Below is the 16 column array of values from file %s
\n\n",random_file_access);
fseek(ptr1,ptr_offset,0);
fread(array,1,random_length,ptr1);
for(row=0;row<random_row;row++)
    {
    for(col=0;col<incol;col++)
{
if(col%16==0)
                printf("\n");
printf("%5u",array[row][col]);
}
    }
```

```
for(col=0;col<random_col;col++)
    {
    if(col%16==0)
printf("\n");
    printf("%5u",array[random_row][col]);
    }
printf("\n\n");
fclose(ptr1);
printf("Do you wish to look at another area, y or n? \n");
scanf(" %c",&random);
}
    else
{
printf("Maximum number of allowed rows is %d and the
maximum ",inrow);
printf("number of allowed columns is %d \n",incol);
}
    }
printf("\nDo you wish to change something, y or n? \n");
scanf(" %c",&program);
}
}

sort(list,size)
unsigned char list[];
unsigned size;                                  /*SORTING FUNCTION*/
{
int out,in,temp1;
for(out=0;out<size-1;out++)
    {
    for(in=out+1;in<size;in++)
        {
if(list[out]>list[in])
    {
    temp1=list[in];
    list[in]=list[out];
    list[out]=temp1;
    }
}
    }
return;
}
```

Appendix C. *Program PREPROC.C*

This program is similar to *log3x3.c*, however, it is designed to preprocess multiple files without user intervention. The program consists of a reduction routine, median filter, normalization, and threshoiding options. Upon entering the program, the user is ask if the data files are test or template files. The only difference lies in the names of the input and output files. For a test file, the input filename may be 2tst1 for example, while a template file would be named 2deg1. The output files are appended with .256 (i.e. 2tst1.256). The 2 represents either the angle of arrival(AOA) for the template file or the number of the test file. The 1 represents the length of the fiber.

The program next prompts the user for the initial and ending AOA for template files, or initial and ending number for test files. For example, if the user has taken *data for AOA from 3° to 15° in one* degree increments, then he enters 3 and 15 for initial and ending AOA respectively. He then enters the length of the fiber (i.e. 2 for a two inch fiber) at the next prompt, and the program will automatically preprocess files 3deg2 through 15deg2. It will locate and preprocess these files according to the information the user supplies. The options are log compression, a histogram to help determine threshold, threshold value, binarization, and input and output directory. The program offers an input and output directory suggestion, and the user can select this by typing 'a', or the user can enter his own input and output directory. The program uses a SWITCH statement to do this so that other directories may be added to the program at any time just by appending them to the SWITCH statement in the source code. This will help save typing the entire directory path each time.

In addition to the preprocessed data output files (which are appended with .256) there is a file called TEMPHIST.DOC for template files or TESTHIST.DOC for test files which is always placed in the TEMPLATE or TEST directory respectively. This file contains the peak value before and after normalization as well as

it's coordinates, the sum of the intensities, and a histogram for each data file. The beginning of this file indicates what program options were selected.

In summary, the program reduces each 512 x 512 file to a 256 x 256 file. It also executes multiple files automatically. The files must be in one degree increments for template files and one number increments for test files. For example, using a one inch fiber, the user may take test data at 3°, 19°, and 12°, however, he would name these files 0tst1, 1tst1, and 2tst1 respectively. When all processing is completed (i.e. the execution of *2dfft*, *template.c*, and *distance.c*), the program *distance.c* will classify the test files, hopefully correctly. The program also requires expanded memory to run since it reads and writes to temporary files in drive D. Each file takes approximately 1 minute and 20 seconds to preprocess.

```
/*Program PREPROC.C*/

/*Written by Capt John W. Welker*/



/*This program converts a 512 x 512 spiricon file into a 256 x 256*/
/*file by grabbing only the center 256 x 256 pixels in the spiricon*/
/*file.                                                            */
/*This program performs the pre-processing on a Spiricon image file*/
/*This pre-processing consists of a median filter operation which*/
/*removes spikes in the data. The data is then normalized to the */
/*maximum of 255. The data may be log compressed to increase the*/
/*lower intensities. A histogram of the data can then be examined*/
/*to determine the best threshold. Data below the threshold is*/
/*set to zero. Data above the threshold is either set to 255.*/
/*or left unchanged. Recursion means that the median value is*/
/*placed into the middle of the 3 x 3 matrix before it calculates*/
/*the next median value.*/


/*COMPILER DIRECTIVES*/
#include <stdio.h>
#include <string.h>
#include <math.h>

/*DEFINE MACROS*/
#define inrow 64
#define incol 512

#define slice 4

#define outrow 66
#define outcol 256

/*DECLARE POINTERS TO FILE STRUCTURE*/
FILE *ptr_in,*temp_ptr,*temp1_ptr,*ptr_out,*histogram_ptr;

/*START MAIN PROGRAM*/
void main()
```

```
{

/*DECLARE LOCAL VARIABLES*/
unsigned char filename_in[256],filename_out[256],temp_file[256];
unsigned char temp1_file[256],directory_in[256],directory_out[256];
unsigned char histogram_file[256],temp[9];
unsigned char array_in[inrow][incol],array_out[outrow][outcol];
unsigned long length_in=0,length=0,sum=0,num_byte=0,peakbyte=0;
unsigned long slice_sum=0,norm_sum=0;
unsigned long segment1,segment2,segment3,segment4,segment5,segment6;
unsigned long segment7,segment8,segment9,segment10,segment11;
unsigned long segment12,segment13,segment14,segment15,segment16;
unsigned row,col,count,count1,count2,count3,c,r,a=0,max=0,peakmax=0;
unsigned peakrow=0,peakcol=0,begin_degree=0,end_degree=0;
unsigned fiber_length=0,begin_test=0,end_test=0,norm_max=0;
unsigned threshold=0;
char test,log_scale,histo,thresh,binary;
float norm_scale=0.0;

length_in=inrow*incol;
length=outrow*outcol;

strcpy(temp_file,"d:\\temp");
strcpy(temp1_file,"d:\\temp1");


printf("\n\nAre these test files, y or n?     ");
scanf(" %c",&test);
printf("\n\n");
if(test=='y')
    {
    printf("What is the number of your initial test file?     ");
    scanf("%u",&begin_test);
    printf("\n\n");
    printf("What is the number of your last test file?     ");
    scanf("%u",&end_test);
    printf("\n\n");
    begin_degree=begin_test;
    end_degree=end_test;
    strcpy(histogram_file,"c:\\spir\\test\\testhist.doc");
    }
else
```

```
    {
    printf("Your first input file corresponds to what angle of
    incidence?     ");
    scanf("%u",&begin_degree);
    printf("\n\n");
    printf("Your last input file corresponds to what angle of
    incidence?     ");
    scanf("%u",&end_degree);
    printf("\n\n");
    strcpy(histogram_file,"c:\\spir\\template\\temphist.doc");
    }
printf("How long is your fiber?     ");
scanf("%u",&fiber_length);
printf("\n\n");
printf("Do you wish to perform a log compression of the
data, y or n?     ");
scanf(" %c",&log_scale);
printf("\n\n");
if(log_scale=='y')
    fprintf(histogram_ptr,"A log compression was performed on this
    data. \n\n");
printf("Do you wish to see a histogram of the data ");
printf("for thresholding purposes, y or n?     ");
scanf(" %c",&histo);
printf("\n\n");
printf("Do you wish to set a threshold, y or n?     ");
scanf(" %c",&thresh);
printf("\n\n");
if(thresh=='y')
    {
    printf("Enter threshold value (255 is peak).     ");
    scanf("%u",&threshold);
    printf("\n\n");
    fprintf(histogram_ptr,"Threshold was set at %u \n\n",threshold);
    printf("Do you wish to set intensities above the threshold ");
    printf("to a single value, y or n? \n");
    scanf(" %c",&binary);
    if(binary=='y')
{
printf("Everything below %u will be zero ",threshold);
printf("and everything above or equal to %u will equal
255 \n\n",threshold);
```

```
fprintf(histogram_ptr,"This data was binarized \n\n");
}
    else
printf("Everything below %u will be zero \n",threshold);
    }

/*ENTER FILES AND VARIABLE VALUES FROM KEYBOARD*/
printf("\nEnter directory path of input file or: \n\n");
printf("Enter 'a' for c:\\spir\\rawdata\\     ");
scanf("%s",directory_in);
printf("\n\n");
switch (*directory_in)
    {
    case 'a':
strcpy(directory_in,"c:\\spir\\rawdata\\");
     break;
    }

printf("\nEnter directory path of output file or: \n\n");
printf("Enter 'a' for c:\\spir\\procdata\\     ");
scanf("%s",directory_out);
printf("\n\n");
switch (*directory_out)
    {
    case 'a':
strcpy(directory_out,"c:\\spir\\procdata\\");
     break;
    }

printf("Opening output file %s \n\n",histogram_file);
if( (histogram_ptr=fopen(histogram_file,"wb"))==NULL)
    {
    printf("Can't open output file %s \n",histogram_file);
    exit();
    }

for(count=begin_degree;count<=end_degree;count++)
{

if(test=='y')
    {
    sprintf(filename_in,"%s%utst%u",directory_in,count,fiber_length);
```

```
    sprintf(filename_out,"%s%utst%u.256",directory_out,count,
    fiber_length);
    }
else
    {
    sprintf(filename_in,"%s%udeg%u",directory_in,count,fiber_length);
    sprintf(filename_out,"%s%udeg%u.256",directory_out,count,
    fiber_length);
    }

/*OPEN INPUT AND OUTPUT FILES*/
printf("\nOpening input file %s \n",filename_in);
if( (ptr_in=fopen(filename_in,"rb"))==NULL)
    {
    printf("Can't open input file %s \n",filename_in);
    exit();
    }
printf("Opening output file %s \n\n",temp_file);
if( (temp_ptr=fopen(temp_file,"wb"))==NULL)
    {
    printf("Can't open output file %s \n",temp_file);
    exit();
    }
fseek(ptr_in,65538,0);
for(count1=0;count1<slice;count1++)
    {
    printf("reading slice #%u \n\n",count1+1);
    fread(array_in,1,length_in,ptr_in);
    for(row=0;row<inrow;row++)
{
for(col=0;col<incol;col++)
    {
    if(col>127 && col<384)
                {
array_out[row][col-128]=array_in[row][col];
}
            }
}
    printf("\n\nWriting slice #%u \n\n",count1+1);
    fwrite(array_out,1,length-2*outcol,temp_ptr);
    }
fclose(ptr_in);
```

```
fclose(temp_ptr);


      /*BEGIN MEDIAN FILTER PROCESSING*/


/*OPEN INPUT AND OUTPUT FILES*/
printf("\nOpening input file %s \n",temp_file);
if( (temp_ptr=fopen(temp_file,"rb"))==NULL)
    {
    printf("Can't open input file %s \n",temp_file);
    exit();
    }
printf("Opening output file %s \n\n",temp1_file);
if( (temp1_ptr=fopen(temp1_file,"wb"))==NULL)
    {
    printf("Can't open output file %s \n",temp1_file);
    exit();
    }

sum=0;
max=0;
num_byte=0;
peakmax=0;
peakbyte=0;
peakrow=0;
peakcol=0;

for(count1=0;count1<slice;count1++)
    {
    if(count1==0)                                   /*FIRST PASS ONLY*/
{
printf("reading slice #%u \n",count1+1);
    /*FIRST READ INTO ARRAY*/
fread(array_out,1,length-2*outcol,temp_ptr);
printf("temp_ptr = %lu \n\n",ftell(temp_ptr));
r=0;
for(count2=0;count2<outrow-4;count2++)
 /*FIRST MEDIAN FILTER LOOP*/
    {
    c=0;
    for(count3=0;count3<outcol-2;count3++)
```

```
{
a=0;
for(row=r;row<r+3;row++)       /*SELECT VALUES FOR*/
    {                              /*3 X 3 ARRAY*/
    for(col=c;col<c+3;col++)/*AND PREPARE FOR SORT*/
{
temp[a]=array_out[row][col];
a++;
}
    }
sort(temp,a);                  /*SORT  3 X 3 ARRAY*/
c++;
array_out[r+1][c]=temp[a/2];/*PLACE MEDIAN VALUE IN*/
}                             /*CENTER OF 3 X 3 ARRAY*/
    r++;
    }
for(col=0;col<outcol;col++)           /*SET FIRST ROW EQUAL*/
    array_out[0][col]=array_out[1][col];    /*TO SECOND ROW*/
for(row=0;row<outrow-4;row++)
    {                                /*RESET FIRST AND LAST*/
    array_out[row][0]=array_out[row][1];     /*COLUMN*/
    array_out[row][outcol-1]=array_out[row][outcol-2];
    }
printf("Calculating first slice intensity and maximum \n\n");
slice_sum=0;
for(row=0;row<outrow-4;row++)     /*CALCULATE MAXIMUM VALUE*/
    {                                   /*AND INTENSITY*/
    for(col=0;col<outcol;col++)
{
sum+=array_out[row][col];
slice_sum+=array_out[row][col];
num_byte++;
max =
(max > array_out[row][col]) ?max:array_out[row][col];
if(peakmax != max)
    {
    peakmax=max;
    peakbyte=num_byte;
    }
}
    }
printf("The sum of this slice is %lu \n\n",slice_sum);
```

```
printf("Writing slice #%u \n",count1+1);
fwrite(array_out,1,length-4*outcol,temp1_ptr);
/*WRITE ALL BUT LAST TWO*/
/*ROWS TO OUTPUT FILE*/
printf("temp1_ptr = %lu \n\n",ftell(temp1_ptr));
}
    else                                        /*BEGIN ALL SUCCESSIVE*/
{                                       /*PASSES THROUGH FILTER*/
for(row=0;row<2;row++)
    {
    for(col=0;col<outcol;col++)     /*LAST TWO ROWS ARE NOW*/
{                                  /*FIRST TWO ROWS*/
array_out[row][col]=array_out[outrow-4+row][col];
}
    }
printf("reading slice #%u \n",count1+1);
fread(&array_out[2][0],1,length-2*outcol,temp_ptr);
/*READ IN SECOND SLICE OF*/
/*DATA STARTING IN THIRD*/
     /*ROW OF ARRAY*/
printf("temp_ptr = %lu \n\n",ftell(temp_ptr));
r=0;
for(count2=0;count2<outrow-2;count2++)
    {                              /*SUCCESSIVE MEDIAN FILTER*/
    c=0;                                    /*LOOPS*/
    for(count3=0;count3<outcol-2;count3++)
{
a=0;
for(row=r;row<r+3;row++) /*SELECT VALUES FOR 3 X 3*/
    {                         /*ARRAY AND PREPARE FOR*/
    for(col=c;col<c+3;col++)      /*SORTING*/
{
temp[a]=array_out[row][col];
a++;
}
    }
sort(temp,a);                     /*SORT 3 X 3 ARRAY*/
c++;
  /*PLACE MEDIAN VALUE INTO*/
                                            /*CENTER OF 3 X 3 ARRAY*/
array_out[r+1][c]=temp[a/2];
```

```
}
    r++;
    }
for(row=0;row<outrow-2;row++)       /*RESET FIRST AND LAST*/
    {                                        /*COLUMNS*/
    array_out[row][0]=array_out[row][1];
    array_out[row][outcol-1]=array_out[row][outcol-2];
    }
if(count1<slice-1)                    /*INTERMEDIATE LOOPS*/
            {
    printf("Calculating intermediate slice intensity and
    maximum \n\n");
    slice_sum=0;
    for(row=0;row<outrow-2;row++)
{
for(col=0;col<outcol;col++)/*CALCULATE MAXIMUM*/
    {                      /*VALUE AND INTENSITY*/
    sum+=array_out[row][col];
    slice_sum+=array_out[row][col];
    num_byte++;
    max =
(max > array_out[row][col]) ?max:array_out[row][col];
    if(peakmax != max)
     {
     peakmax=max;
     peakbyte=num_byte;
     }
    }
}
    printf("The sum of this slice is %lu \n\n",slice_sum);
    printf("Writing slice #%u \n",count1+1);
    fwrite(array_out,1,length-2*outcol,temp1_ptr)
  /*WRITE TO OUTPUT FILE*/
  /*DURING INTERMEDIATE*/
  /*PASSES THROUGH FILTER*/
    printf("temp1_ptr = %lu \n\n",ftell(temp1_ptr));
    }
else
    {                             /*LAST PASS THROUGH FILTER*/
     /*RESET FIRST AND LAST COLUMN*/
    array_out[outrow-2][0]=array_out[outrow-2][1];
    array_out[outrow-2][outcol-1]=
```

```
    array_out[outrow-2][outcol-2];
    for(col=0;col<outcol;col++)    /*RESET LAST ROW*/
array_out[outrow-1][col]=array_out[outrow-2][col];
    printf("Calculating last intensity and maximum \n\n");
    slice_sum=0;
    for(row=0;row<outrow;row++)  /*CALCULATE MAXIMUM VALUE*/
{                          /*AND INTENSITY IN LAST*/
for(col=0;col<outcol;col++)         /*SLICE*/
    {
    sum+=array_out[row][col];
    slice_sum+=array_out[row][col];
    num_byte++;
    max =
(max > array_out[row][col]) ?max:array_out[row][col];
    if(peakmax != max)
     {
     peakmax=max;
     peakbyte=num_byte;
     }
    }
}
    printf("The sum of this slice is %lu \n\n",slice_sum);
    printf("Writing slice #%u \n",count1+1);
    fwrite(array_out,1,length,temp1_ptr);
/*WRITE ALL OF LAST SLICE*/
    /*TO OUTPUT FILE*/
    printf("temp1_ptr = %lu \n\n",ftell(temp1_ptr));
    }
}
    }
fclose(temp_ptr);                                    /*CLOSE FILES*/
fclose(temp1_ptr);
peakrow=(peakbyte/(long)outcol);
peakcol=(peakbyte % (long)outcol)-1;
printf("The sum of the first %lu intensities is %lu \n\n",
num_byte,sum);
fprintf(histogram_ptr,"\n\n\n\n\nThe sum of the first %lu intensities
after filtering is %lu \n\n",num_byte,sum);
fprintf(histogram_ptr,"The peak intensity of %u after filtering
occurs at \n",max);
fprintf(histogram_ptr,"row %u and column %u in file %s. \n\n",
peakrow,peakcol,filename_out);
```

```
                    /*BEGINNING NORMALIZATION ROUTINE*/

    norm_sum=0;
    norm_max=0;
    num_byte=0;
    segment1=0;
    segment2=0;
    segment3=0;
    segment4=0;
    segment5=0;
    segment6=0;
    segment7=0;
    segment8=0;
    segment9=0;
    segment10=0;
    segment11=0;
    segment12=0;
    segment13=0;
    segment14=0;
    segment15=0;
    segment16=0;
    if(log_scale=='y')
norm_scale=255.5/log((double)max);
    else
norm_scale=255.5/(float)max;
    printf("Scaling factor is %f \n\n",norm_scale);
    printf("\nOpening input file %s for normalization\n",temp1_file);
    if( (temp1_ptr=fopen(temp1_file,"rb"))==NULL)
{
printf("Can't open input file %s \n",temp1_file);
exit();
}
    printf("Opening output file %s \n\n",filename_out);
    if( (ptr_out=fopen(filename_out,"wb"))==NULL)
{
printf("Can't open input file %s \n",filename_out);
exit();
}
    for(count1=0;count1<slice;count1++)
{
```

```
printf("reading slice #%u \n",count1+1);
fread(array_out,1,length-2*outcol,temp1_ptr);
printf("temp1_ptr=%lu \n\n",ftell(temp1_ptr));
printf("Value of array_out[1][1] before scaling is %u \n",
array_out[1][1]);
for(row=0;row<outrow-2;row++)
    {
    for(col=0;col<outcol;col++)
{
if(log_scale=='y')
    {
    array_out[row][col]=
      (double)norm_scale*(log((double)array_out[row][col]));
    if(row==1 && col==1)
printf("Value of array_out[1][1] after log
compression is %u \n",array_out[1][1]);
    }
else
    array_out[row][col]*=norm_scale;
if(row==1 && col==1)
    printf("Value of array_out[1][1] after scaling
    is %u \n\n",array_out[1][1]);
norm_sum+=array_out[row][col];
norm_max=
(norm_max>array_out[row][col]) ?norm_max:array_out[row][col];
num_byte++;
if(histo=='y')
                    {
    if(array_out[row][col]<16)
segment1++;
    else if(array_out[row][col]<32)
segment2++;
    else if(array_out[row][col]<48)
segment3++;
    else if(array_out[row][col]<64)
segment4++;
    else if(array_out[row][col]<80)
segment5++;
    else if(array_out[row][col]<96)
segment6++;
    else if(array_out[row][col]<112)
segment7++;
```

```
    else if(array_out[row][col]<128)
segment8++;
    else if(array_out[row][col]<144)
segment9++;
    else if(array_out[row][col]<160)
segment10++;
    else if(array_out[row][col]<176)
segment11++;
    else if(array_out[row][col]<192)
segment12++;
    else if(array_out[row][col]<208)
segment13++;
    else if(array_out[row][col]<224)
segment14++;
    else if(array_out[row][col]<240)
segment15++;
    else if(array_out[row][col]<=255)
segment16++;
    }
if(thresh=='y')
    {
    if(binary=='y')
{
if(array_out[row][col]<threshold)
    array_out[row][col]=0;
else
    array_out[row][col]=255;
}
    else
{
if(array_out[row][col]<threshold)
    array_out[row][col]=0;
}
    }
}
    }
printf("Writing slice #%u \n",count1+1);
fwrite(array_out,1,length-2*outcol,ptr_out);
printf("ptr_out=%lu \n\n",ftell(ptr_out));
}
    fprintf(histogram_ptr,"The sum of the first %lu intensities
    after normalization is %lu \n",num_byte,norm_sum);
```

```
    fprintf(histogram_ptr,"The normalized maximum in
    file %s is %u \n\n",filename_out,norm_max);
    if(histo=='y')
{
fprintf(histogram_ptr,"\n\t\t\t\tHISTOGRAM OF DATA \n");
fprintf(histogram_ptr,"\t\t(Value in parentheses is value
on spiricon plot) \n\n\n");
fprintf(histogram_ptr,"%.2f \tpercent of the values lie
between    0(0)         and    15(127). \n",
(float)segment1/(float)num_byte*100.0);
fprintf(histogram_ptr,"%.2f \tpercent of the values lie
between    16(128)      and    31(255). \n",
(float)segment2/(float)num_byte*100.0);
fprintf(histogram_ptr,"%.2f \tpercent of the values lie
between    32(256)      and    47(382). \n",
(float)segment3/(float)num_byte*100.0);
fprintf(histogram_ptr,"%.2f \tpercent of the values lie
between    48(383)      and    63(510). \n",
(float)segment4/(float)num_byte*100.0);
fprintf(histogram_ptr,"%.2f \tpercent of the values lie
between    64(511)      and    79(637). \n",
(float)segment5/(float)num_byte*100.0);
fprintf(histogram_ptr,"%.2f \tpercent of the values lie
between    80(638)      and    95(765). \n",
(float)segment6/(float)num_byte*100.0);
fprintf(histogram_ptr,"%.2f \tpercent of the values lie
between    96(766)      and    111(892). \n",
(float)segment7/(float)num_byte*100.0);
fprintf(histogram_ptr,"%.2f \tpercent of the values lie
between    112(893)     and    127(1020). \n",
(float)segment8/(float)num_byte*100.0);
fprintf(histogram_ptr,"%.2f \tpercent of the values lie
between    128(1021)    and    143(1147). \n",
(float)segment9/(float)num_byte*100.0);
fprintf(histogram_ptr,"%.2f \tpercent of the values lie
between    144(1148)    and    159(1275). \n",
(float)segment10/(float)num_byte*100.0);
fprintf(histogram_ptr,"%.2f \tpercent of the values lie
between    160(1276)    and    175(1402). \n",
(float)segment11/(float)num_byte*100.0);
fprintf(histogram_ptr,"%.2f \tpercent of the values lie
between    176(1403)    and    191(1530). \n",
```

```
(float)segment12/(float)num_byte*100.0);
fprintf(histogram_ptr,"%.2f \tpercent of the values lie
between    192(1531)    and    207(1657). \n",
(float)segment13/(float)num_byte*100.0);
fprintf(histogram_ptr,"%.2f \tpercent of the values lie
between    208(1658)    and    223(1785). \n",
(float)segment14/(float)num_byte*100.0);
fprintf(histogram_ptr,"%.2f \tpercent of the values lie
between    224(1786)    and    239(1912). \n",
(float)segment15/(float)num_byte*100.0);
fprintf(histogram_ptr,"%.2f \tpercent of the values lie
between    240(1913)    and    255(2040). \n",
(float)segment16/(float)num_byte*100.0);
}

    fclose(temp1_ptr);
    fclose(ptr_out);

}
fclose(histogram_ptr);

}

sort(list,size)
unsigned char list[];
unsigned size;                                  /*SORTING FUNCTION*/
{
int out,in,temp1;
for(out=0;out<size-1;out++)
    {
    for(in=out+1;in<size;in++)
        {
if(list[out]>list[in])
    {
    temp1=list[in];
    list[in]=list[out];
    list[out]=temp1;
    }
}
    }
return;
}
```

Appendix D. *Program TEMPLATE.C*

This program is used to extract the low frequency Fourier components (49 numbers) and create a file containing feature vectors, each of which represents the shape of the output intensity pattern for each degree of incidence. The inputs to this program are files which were previously Fourier transformed and have the FT prefix (i.e. FT15deg3.256). The FT indicates that the file was previously Fourier transformed. The 15 represents a 15° angle of arrival (AOA), deg is indicative of a template file, the 3 is for a 3 inch length of fiber, and the 256 shows that the file was preprocessed with the program *preproc.c*. More information on input information, file names, directory paths, and preprocessing can be found in appendix C.

The program places the output data (i.e. feature vectors) in the file TEMPLATE.DAT or TEST.DAT in directory TEMPLATE or TEST respectively. Each row in this file represents a particular AOA and contains 49 columns. This file is strictly used for further processing by the program *distance.c*. Another output file, TEMPLATE.DOC or TEST.DOC, contains the same information but can be viewed on the screen or sent to a printer. These files are also contained in the directories TEMPLATE and TEST respectively.

In summary, the output files are variable length. The file TEMPLATE.DAT or TEST.DAT is always 49 columns wide, but the number of rows depends on the number of files processed. This information is needed for the program *distance.c*.

```
/*Program TEMPLATE.C*/

/*Written by Capt John W. Welker*/



/*This program places the fundamental and first three harmonics*/
/*from one or several 256 x 256 fourier transform files into a 49*/
/*column array where each row represents a 49 component feature*/
/*vector. The files must be named*/
/*[ft(degrees number)deg(fiber length).256] Example: ft10deg3.256.*/
/*This is the fourier transform file representing*/
/*an angle of incidence of 10 degrees and a fiber length of 3*/
/*inches. The number of your initial test file refers to the*/
/*number preceeding tst in the first test file*/
/*filename[ft0tst(fiber length).256] and then*/
/* number them consecutively*/
/* You have your choice of input and output directory*/


/*COMPILER DIRECTIVES*/
#include <stdio.h>
#include <string.h>
#include <math.h>

/*DEFINE MACROS*/
#define inrow 7
#define incol 256

/*DECLARE POINTERS TO FILE STRUCTURE*/
FILE *ptr_in,*template_data_ptr,*template_doc_ptr;

/*START MAIN PROGRAM*/
void main()
{

/*DECLARE LOCAL VARIABLES*/
unsigned char filename_in[256],filename_data_out[256];
unsigned char filename_doc_out[256],vector[49];
unsigned char directory_in[256],directory_out[256];
```

```
unsigned char array_in[inrow][incol];
unsigned long length_in=0;
unsigned row,col,count,a=0;
unsigned begin_degree=0,end_degree=0,begin_test=0,end_test=0;
unsigned fiber_length=0;
char test;

length_in=inrow*incol;

/*ENTER FILES AND VARIABLE VALUES FROM KEYBOARD*/
printf("\n\nAre these test files, y or n?     ");
scanf(" %c",&test);
printf("\n\n");
if(test=='y')
    {
    printf("What is the number of your initial test file?     ");
    scanf("%u",&begin_test);
    printf("\n\n");
    printf("What is the number of your last test file?     ");
    scanf("%u",&end_test);
    printf("\n\n");
    begin_degree=begin_test;
    end_degree=end_test;
    }
else
    {
    printf("Your first fourier transform input file corresponds to");
    printf(" what angle of incidence?     ");
    scanf("%u",&begin_degree);
    printf("\n\n");
    printf("Your last fourier transform input file corresponds to
    what");
    printf(" angle of incidence?     ");
    scanf("%u",&end_degree);
    printf("\n\n");
    }
printf("How long is your fiber?     ");
scanf("%u",&fiber_length);
printf("\n\n");

printf("\nEnter directory path of input file or: \n\n");
printf("Enter 'a' for c:\\spir\\ftdata\\     ");
```

```
scanf("%s",directory_in);
printf("\n\n");
switch (*directory_in)
    {
    case 'a':
strcpy(directory_in,"c:\\spir\\ftdata\\");
     break;
    }
if(test=='y')
    {
    printf("\nEnter directory path of output file or: \n\n");
    printf("Enter 'a' for c:\\spir\\test\\     ");
    scanf("%s",directory_out);
    printf("\n\n");
    switch (*directory_out)
{
case 'a':
    strcpy(directory_out,"c:\\spir\\test\\");
    break;
}
    sprintf(filename_data_out,"%stest.dat",directory_out);
    sprintf(filename_doc_out,"%stest.doc",directory_out);
    }
else
    {
    printf("\nEnter directory path of output file or: \n\n");
    printf("Enter 'a' for c:\\spir\\template\\     ");
    scanf("%s",directory_out);
    printf("\n\n");
    switch (*directory_out)
{
case 'a':
    strcpy(directory_out,"c:\\spir\\template\\");
    break;
}
    sprintf(filename_data_out,"%stemplate.dat",directory_out);
    sprintf(filename_doc_out,"%stemplate.doc",directory_out);
    }

/*OPEN OUTPUT FILES*/
printf("Opening output file %s \n\n",filename_data_out);
if( (template_data_ptr=fopen(filename_data_out,"wb"))==NULL)
```

```
    {
    printf("Can't open output file %s \n",filename_data_out);
    exit();
    }
printf("Opening output file %s \n\n",filename_doc_out);
if( (template_doc_ptr=fopen(filename_doc_out,"wb"))==NULL)
    {
    printf("Can't open output file %s \n",filename_doc_out);
    exit();
    }
for(count=begin_degree;count<=end_degree;count++)
    {
    a=0;
    if(test=='y')
{
fprintf(template_doc_ptr,"Test vector %u:     \t",count);
sprintf(filename_in,"%sft%utst%u.256",directory_in,count,
fiber_length);
}
    else
{
fprintf(template_doc_ptr,"Template vector %u: \t",count);
sprintf(filename_in,"%sft%udeg%u.256",directory_in,count,
fiber_length);
}

/*OPEN INPUT FILE*/
    printf("\n\nOpening input file %s \n\n",filename_in);
    if( (ptr_in=fopen(filename_in,"rb"))==NULL)
{
printf("Can't open input file %s \n",filename_in);
exit();
}
    fseek(ptr_in,32000,0);
    fread(array_in,1,length_in,ptr_in);
    for(row=0;row<inrow;row++)
{
for(col=0;col<incol;col++)
    {
    if(col>124 && col<132)
                {
vector[a]=array_in[row][col];
```

```
printf("%3u ",vector[a]);
fprintf(template_doc_ptr,"%u,",vector[a]);
if(a==12||a==25||a==38)
    fprintf(template_doc_ptr,"\n\t\t\t");
a++;
}
    }
printf("\n");
}
    fprintf(template_doc_ptr,"\n\n");
    printf("\nWriting vector #%u \n\n",count),
    fwrite(vector,1,a,template_data_ptr);
    fclose(ptr_in);
    }
fclose(template_data_ptr);
fclose(template_doc_ptr);
}
```

Appendix E. *Program DISTANCE.C*

This program calculates distances between template vectors in the file TEMPLATE.DAT to verify adequate separation. It can also calculate the distances between the test vectors in file TEST.DAT and the template vectors. In this case the minimum distance classifies each test vector. These distances are placed in the file TEMPDIST.DOC or TESTDIST.DOC in the TEMPLATE or TEST directory respectively. TESTDIST.DOC also contains the estimated angle of arrival (i.e. classification) for each test vector.

The program only needs to know which distances are required (i.e. distances *between* template vectors or distances between test and template vectors) and how many vectors (i.e. number of rows) are contained in each file. The user can also choose the input directory, which should be either TEMPLATE or TEST and TEMPLATE.

```
/*Program DISTANCE.C*/

/*Written by Capt John W. Welker*/



/*This program calculates distances between feature vectors.*/
/*You have the option of calculating distances between the template*/
/*vectors in the file TEMPLATE.DAT to determine if your template*/
/*vectors are adequately separated in your feature space, or you*/
/*can calculate the distances between test vectors in the*/
/*file TEST.DAT and the template vectors in the file TEMPLATE.DAT.*/

/*COMPILER DIRECTIVES*/
#include <stdio.h>
#include <string.h>
#include <math.h>

/*DECLARE POINTERS TO FILE STRUCTURE*/
FILE *test_ptr,*template_ptr,*distance_ptr;

/*START MAIN PROGRAM*/
void main()
{

/*DECLARE LOCAL VARIABLES*/
unsigned char test_filename[256],template_filename[256];
unsigned char distance_filename[256];
unsigned char test_directory[256],template_directory[256];
unsigned char test_vector[49],template_vector[49];
unsigned count1,count2,a,min_vector=0;
unsigned num_test_rows=0,num_template_rows;
double sum=0.0,distance=0.0,min_distance=0.0;
char test;

/*ENTER FILES AND VARIABLE VALUES FROM KEYBOARD*/
printf("\n\nAre you comparing test vectors to the template
vectors, y or n?     ");
scanf(" %c",&test);
printf("\n\n");
```

```
printf("How many vectors are in your template file?     ");
scanf("%u",&num_template_rows);
printf("\n\n");
if(test=='y')
    {
    printf("How many vectors are in your test file?     ");
    scanf("%u",&num_test_rows);
    printf("\n\n");
    }
else
    {
    num_test_rows=num_template_rows;
    }

printf("\nEnter directory path of template file or: \n\n");
printf("Enter 'a' for c:\\spir\\template\\     ");
scanf("%s",template_directory);
printf("\n\n");
switch (*template_directory)
    {
    case 'a':
strcpy(template_directory,"c:\\spir\\template\\");
     break;
    }

sprintf(template_filename,"%stemplate.dat",template_directory);

/*OPEN TEMPLATE FILE*/
printf("Opening template file %s \n\n",template_filename);
if( (template_ptr=fopen(template_filename,"rb"))==NULL)
    {
    printf("Can't open output file %s \n",template_filename);
    exit();
    }

if(test=='y')
    {
    printf("\nEnter directory path of test file or: \n\n");
    printf("Enter 'a' for c:\\spir\\test\\     ");
    scanf("%s",test_directory);
    printf("\n\n");
    switch (*test_directory)
```

```
{
case 'a':
    strcpy(test_directory,"c:\\spir\\test\\");
    break;
}
    sprintf(test_filename,"%stest.dat",test_directory);
    sprintf(distance_filename,"%stestdist.doc",test_directory);

    /*OPEN TEST FILE*/
    printf("Opening test file %s \n\n",test_filename);
    if( (test_ptr=fopen(test_filename,"rb"))==NULL)
{
printf("Can't open input file %s \n",test_filename);
exit();
}
    }
else
    {
    sprintf(test_filename,"%stemplate.dat",template_directory);
    sprintf(distance_filename,"%stempdist.doc",template_directory);

    /*OPEN TEST FILE*/
    printf("Opening test file %s \n\n",test_filename);
    if( (test_ptr=fopen(test_filename,"rb"))==NULL)
{
printf("Can't open input file %s \n",test_filename);
exit();
}
    }


/*OPEN DISTANCE FILE*/
printf("Opening distance file %s \n\n",distance_filename);
if( (distance_ptr=fopen(distance_filename,"wb"))==NULL)
    {
    printf("Can't open input file %s \n",distance_filename);
    exit();
    }


for(count1=0;count1<num_test_rows;count1++)
    {
```

```
    fread(test_vector,1,49,test_ptr);
    fseek(template_ptr,0,0);
    min_distance=5.0E100;
    for(count2=0;count2<num_template_rows;count2++)
{
fread(template_vector,1,49,template_ptr);
sum=0.0;
distance=0.0;
for(a=0;a<49;a++)
    {
    sum+=
pow(((double)test_vector[a]-(double)template_vector[a]),2.0);
    }
distance=sqrt(sum);
if(distance!=0.0)
min_distance = (min_distance < distance) ? min_distance : distance;
if(min_distance==distance)
    min_vector=count2;
}
    if(test=='y')
{
printf("Test vector %u arrived at %u degrees \n\n",
count1,min_vector);
fprintf(distance_ptr,"Test vector %u arrived at %u
degrees    ",count1,min_vector);
fprintf(distance_ptr,"(Distance = %f) \n\n",min_distance);
}
    else
{
printf("Pass %u \n",count1+1);
fprintf(distance_ptr,"The minimum distance of %f occurs ",
min_distance);
fprintf(distance_ptr,"between vector %u and vector %u \n\n",
count1,min_vector);
}
    }
fclose(test_ptr);
fclose(template_ptr);
fclose(distance_ptr);
}
```

Appendix F. *Program 2562SPIR.C*

This program is used to expand a 256 x 256 data file to a 512 x 512 Spiricon file for viewing by the Spiricon system. The user only needs to enter a directory path and filename without extension. The program automatically appends .256 to the end of the input filename. The output file from this program is appended with a .512 to indicate that it is a 512 x 512 file. It expands the file by zero filling, inserts the two byte header, and then appends the SPIR.FIG file to the end so that Spiricon will recognize it. The program expands the file within a few seconds.

```
/*Program 2562SPIR.C*/

/*Written by Capt John W. Welker*/



/*This program converts a 256 x 256 file previously reduced from a*/
/*512 x 512 spiricon file back into a 512 x 512 spiricon file.*/




#include <stdio.h>
#include <string.h>
#include <math.h>

#define inrow 64
#define incol 256

#define slice 8

#define outrow 64
#define outcol 512

FILE *ptr1,*ptr2,*ptr3;

void main()
{

unsigned char filename[256],filename_in[256],filename_out[256];
unsigned char array_in[inrow][incol],array_out[outrow][outcol];
unsigned long length_in=0,length_out=0;
unsigned row,col,count1;
char header[] = {'\x39' , '\x30'};

length_in=inrow*incol;
length_out=outrow*outcol;

printf("\n\nEnter directory path and filename of input file without
```

```
extension: \n\n");
scanf("%s",filename);
sprintf(filename_in,"%s.256",filename);
sprintf(filename_out,"%s.512",filename);
printf("Opening input file %s \n",filename_in);
ptr1=fopen(filename_in,"rb");
printf("Opening output file %s \n\n",filename_out);
ptr2=fopen(filename_out,"wb");
fwrite(header,1,2,ptr2);
for(count1=0;count1<slice;count1++)
    {
    printf("reading slice #%u \n\n",count1+1);
    if(count1>1 && count1<6)
fread(array_in,1,length_in,ptr1);
    for(row=0;row<outrow;row++)
{
for(col=0;col<outcol;col++)
    {
            if(count1>1 && count1<6)
{
if(col>127 && col<384)
    array_out[row][col]=array_in[row][col-128];
        else
    array_out[row][col]=0;
}
    else
array_out[row][col]=0;
            }
}
    printf("Writing slice #%u \n\n",count1+1);
    fwrite(array_out,1,length_out,ptr2);
    }
printf("Opening input file c:\\spir\\spir.fig \n",filename);
ptr3=fopen("c:\\spir\\spir.fig","rb");
fread(array_out,1,906,ptr3);
fwrite(array_out,1,906,ptr2);
fclose(ptr1);
fclose(ptr2);
fclose(ptr3);
}
```

Bibliography

1. Cole, Capt Geno. *Angle of Arrival Detector.* Patent Application: AF18159. Air Force Weapons Laboratory, Kirtland AFB NM, 1988.
2. Gaskill, Jack D. *Linear Systems, Fourier Transforms, and Optics.* New York: John Wiley & Sons, 1978.
3. Ginsburg, Arthur P. *Visual Information Processing Based on Spatial Filters Constrained by Biological Data.* Doctoral Dissertation. AMRL - TR - 78 -129, Wright- Patterson AFB OH, December 1978.
4. Hecht, Eugene. *Optics.* Reading MA: Addison-Wesley Publishing Company, 1987.
5. Holl, Herbert B. *Angle of Arrival Meter.* Patent Application: S.N. 349128. Unknown, 1982.
6. Kabrisky, Matthew E. Class lecture in EENG 621, Pattern Recognition II. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, April 1989.
7. Keiser, Gerd. *Optical Fiber Communications.* New York: McGraw Hill Book Company, 1983.
8. Letourneau, Capt George B. *Aspect Angle Invariant Synthetic Aperture Radar Pattern Recognition Using the Hough Transform.* MS thesis, AFIT/GE/ENG/89D. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989. (DTIC number not available at this time).
9. *Optical Sensor With High Directional Resolution.* SBIR Phase I Report, Topic AF88-184. Interactive Intelligent Imagery Corporation, Menlo Park CA, March 1989.
10. Parker, Jack H. Jr. *Common Opto-Electronic Laser Detection System (COLDS).* Final Report, July 1985 - August 1985. AFWAL - TR - 86 - 1018, Wright- Patterson AFB OH, December 1986 (AD - B113557).

Vita

John W. Welker [redacted] He moved to Columbia, South Carolina in November 1978 and entered the University of South Carolina in January 1980. In December 1983, he enrolled in the College Senior Engineering Program(CSEP) of the United States Air Force. After receiving a Bachelor of Science degree in Electrical Engineering in December 1984, he attended Officer Training School and was commissioned a Second Lieutenant, 5 April 1985. Following a three year assignment at Edwards Air Force Base, where he performed Defensive Avionics testing for the B1-B, he entered the Electro-optics program at the Air Force Institute of Technology(AFIT). [redacted]

[redacted]

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GE/ENG/89D - 57	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENP	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code): Air Force Institute of Technology (AFIT) (AU) WPAFB, OH 45433 - 6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING SPONSORING ORGANIZATION: Air Force Weapons Lab (WL); 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code): WL/NTCET Kirtland AFB, NM 87117 - 6008	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

ANGLE OF ARRIVAL DETECTION THROUGH ANALYSIS OF OPTICAL FIBER INTENSITY PATTERNS(UNCLASSIFIED)

12. PERSONAL AUTHOR(S)

John W. Welker, Captain, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS thesis	FROM _____ TO _____	1989, December	107

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Angle of Arrival, Optical Fibers, Pattern Recognition
12	09		
20	06	01	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Advisor: Maj S. Rogers, PhD
Associate Professor, Dept of Electrical and Computer Engineering

This thesis examined the feasibility of using optical fibers for angle of arrival (AOA) detection. Specifically, a uniform amplitude plane wave was transmitted through a 3mm diameter, multimode, step - index, plastic fiber. Fiber lengths of 1in, 2in, and 3in were investigated. The output intensity pattern from each optical fiber was detected by a 512 X 512 resolution, charge injection device (CID) camera. The data array from the camera was processed with a median filter, and normalized to the peak intensity. A fast Fourier transform converted the data to the frequency domain where the fundamental and first three harmonic frequencies were extracted. These 49 numbers represented components of a feature vector representing the pattern related to a specific AOA. Twenty - six template vectors, each representing a 1 degree increment from 0 degrees to 25 degrees, were created. Test

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Steven K. Rogers, Associate Professor	22b. TELEPHONE (Include Area Code): (513) 255 - 9266 / 22c. OFFICE SYMBOL: AFIT/ENG

DD Form 1473, JUN 86 *Previous editions are obsolete.*

19. vectors were compared to template vectors using Euclidean distance. The minimum distance classified the test vector AOA.